Apprentissage autonome appliqué au domaine de la robotique

Ing. J. BEAUDART
PIERRARD – Virton

Ce travail explique comment implémenter dans un robot une capacité d'apprentissage telle que celle d'un enfant de 4 mois. Le principe est basé sur la création de vecteurs composés de 3 parties: un contexte (l'état des capteurs avant qu'une action soit réalisée), une action et enfin un résultat (l'état des capteurs après que l'action ait été réalisée). Le robot apprendra de ces associations comment mouvoir ses différentes parties afin d'atteindre un objectif. Comme un enfant, il apprendra au fur et à mesure quels sont les différents mouvements dont il est capable et, plus important, il apprendra également comment associer différents mouvements basiques pour réaliser des mouvements complexes.

Mots-clés: apprentissage - autonomie - robotique.

This work is explaining how to implement into a robot the same learning ability that a 4 months child has. The principle is based on creating vectors composed of 3 parts: a context (state of the sensors before an action), an action and the result (state of the sensors after the action has been performed). The robot will learn from those associations how to move its parts to reach a specific goal. Like a child, he will learn as to his learning what the different movements he is able to do are, and more importantly he will learn how to associate different basic moves to create complex moves.

Key-words: learning - autonomy - robotics.

1. Introduction

1.1 Contexte du projet

Les avancées en mécatronique supposent que des robots de plus en plus performants soient créés afin d'être utilisés pour les industries les plus à la pointe de la technologie. Malheureusement les robots de notre époque sont « idiots » dans le sens où ils ne sont pas capables d'apprendre de façon générique de leur environnement, comme les humains le font dès leur naissance. Des robots possédant une telle compétence seraient infiniment plus utiles et pourraient être utilisés pour travailler dans des environnements inhospitaliers à l'Homme, tel que le travail en eau profonde pour l'industrie pétrolière. Le but à atteindre avec une telle technologie est la capacité d'adaptation totale, c'est-à-dire qu'un robot entrainé à travailler en eau calme devrait pouvoir s'adapter tout seul à travailler en eau mouvementée où les conditions sont complètement différentes.

1.2 Objectifs du projet

L'objectif de ce projet est la conception d'un algorithme d'apprentissage copiant la faculté d'apprentissage d'un enfant de 4 mois. L'algorithme doit pouvoir, en utilisant des données générées par ordinateur, apprendre et généraliser ces données afin de pouvoir s'adapter à des changements de circonstance/environnement. L'approche utilisera des réseaux neuronaux relativement standards sous la forme de cartes auto-organisatrices. Le travail sera basé sur des recherches philosophiques précédemment établies dans ce domaine (Harold Henry Chaput. *Hierarchical Learning Architecture*, University of Austin, Texas [1]). Le but sera alors de répliquer en code Matlab les idées émises, de les assembler et de les améliorer.

2. Notions préalables

Avant d'entrer dans le vif du sujet il est préférable de s'attarder sur les quelques notions élémentaires à la compréhension de ce travail.

2.1 Les cartes auto-organisatrices (Self-Organizing Maps / SOMs)

Les SOMs (Teuvo Kohonen, 1997) sont des réseaux neuronaux particuliers, qui peuvent avoir différentes formes (feuille, cylindre, toroïde – fig.1) et topologies (rectangulaire, hexagonal – fig.2).

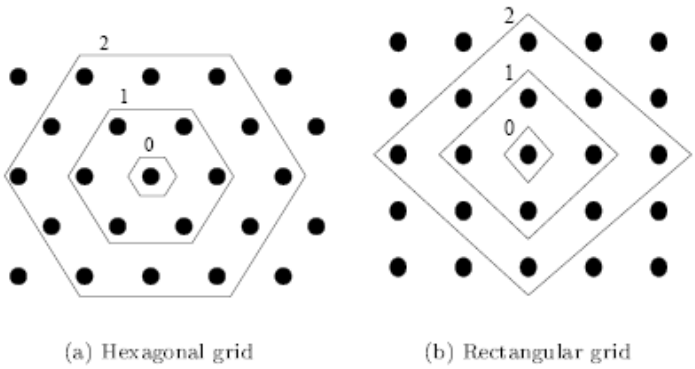


Figure 1: Topologies des SOMs

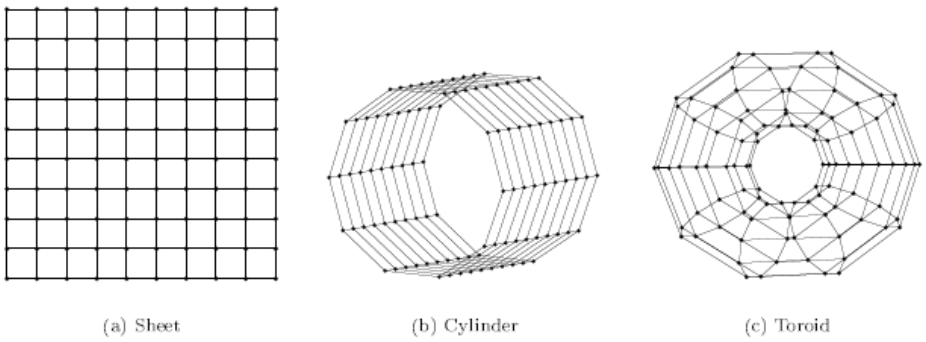


Figure 2: Formes des SOMs

Quelle que soit la forme d'une SOM, un « vecteur de référence » est associé à chaque neurone. Ces vecteurs doivent compter le même nombre d'éléments qu'en comptent les vecteurs de données auxquels ils vont être

confrontés. En effet, chaque donnée va être confrontée à tous les vecteurs de référence afin de déterminer lequel est le plus proche de celui-ci. Le neurone relatif au vecteur de référence le plus proche est alors appelé « neurone vainqueur ».
Il s'en suivra une mise à jour du vecteur référence du neurone vainqueur afin que celui-ci devienne un peu plus proche du vecteur de données. Cette mise à jour a pour but de spécialiser chaque vecteur de référence pour un type de donnée bien précis.

Si « m » = vecteur de référence, « x » = vecteur d'entrée (donnée), et « c » le neurone vainqueur, alors :

$$c = arg\,\min_i ||x - m_i||$$

Formule 1: Détermination du neurone vainqueur

Dans la figure 3, on peut clairement voir l'effet de la mise à jour des vecteurs de référence : les neurones vont se déplacer afin que chacun puisse traiter un certain type de données.

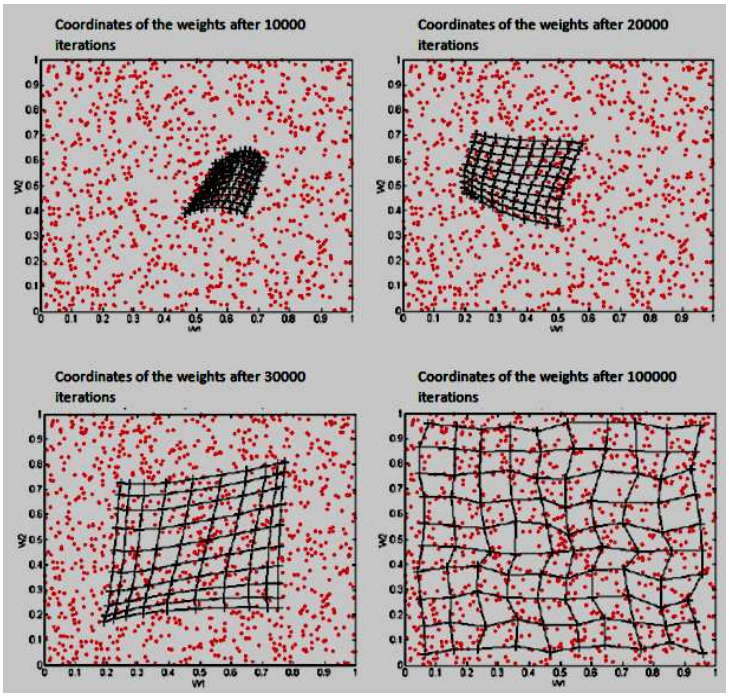


Figure 3 : exemple de mise à jour d'une SOM

2.2 L'architecture dans l'apprentissage

Structurer l'apprentissage avec une architecture est fondamental, c'est d'ailleurs de cette façon que fonctionne notre cerveau. Le principe est de séparer les fonctions et de séparer les niveaux d'intelligence. Séparer les fonctions veut simplement dire que chaque fonction du robot doit être traitée par une SOM différente, de telle façon que le cerveau utilise différentes parties pour traiter les différentes actions et informations (une partie de notre cerveau traite la motricité, une autre la vue etc.). Créer différents niveaux d'intelligence et les séparer de façon distincte est primordial et copie également ce qui se passe dans notre cerveau. Un exemple explicite est celui du dactylographe : en utilisant son niveau d'intelligence le plus bas il ne sait que taper sur des touches, en utilisant le niveau supérieur il sait écrire des mots en faisant des associations de lettres, le niveau supérieur à celui-ci lui permet de faire des phrases en faisant des associations de mots, etc.
Harold Henry Chaput a émis 2 possibilités afin de créer des interconnections entre les différents niveaux : les vecteurs d'activation (qui sont plus adaptés à l'apprentissage du langage et donc qui ne seront pas discutés dans cet article) et les Schémas de Mécanismes de Drescher. Ces schémas de mécanismes sont très simples : chaque entrée (donnée) traitée par le robot doit se présenter sous une forme de vecteur lui-même composé de 3 parties :

- le Contexte : l'état des capteurs avant qu'une action soit réalisée,
- l'action elle-même (représentée par un numéro),
- le Résultat : l'état des capteurs après que l'action ait été réalisée.

Exemple :

EnFaceDeLaPorte / OuvrirLaPorte / PorteOuverte

On peut ensuite transformer ces schémas en « éléments synthétiques » pour être utilisés dans les niveaux d'intelligence plus élevés. Un élément de la partie contexte (et/ou résultat) d'un niveau d'intelligence supérieur peut alors représenter que le robot est dans un état où le fait d'ouvrir la porte aura pour résultat que la porte sera ouverte. Ces transformations en éléments synthétiques sont absolument cruciales dans le processus d'apprentissage comme expliqué plus loin dans cet article.

3. Conception de l'algorithme

3.1 Introduction

L'algorithme peut être découpé en 4 parties principales. Dans l'ordre, celles-ci sont la création et l'initialisation d'une SOM pour chaque action/fonction, l'entrainement et la mise à jour de chacune de ces SOMs, la récolte des vecteurs après l'entrainement et le tri de ces vecteurs et enfin la reconversion de ces vecteurs en schémas susceptibles d'être utilisés par le robot dans un processus de prise de décision. Une ultime étape consiste à introduire les schémas récoltés dans les vecteurs d'entrée du niveau supérieur afin de réaliser les fameuses interconnexions entre les différents niveaux d'intelligence.

Note : Pour une raison de synthétisation, la première partie de la thèse complète n'est pas reprise dans cet article. Cette partie reprend la traduction brute des idées de Chaput, leur assemblement et les résultats des tests effectués sur ces principes basiques. Cet article traite donc les principes améliorés qui ont rendu l'algorithme beaucoup plus performant en termes de vitesse et de résultats.

3.2 Initialisation d'une SOM

Paramètres d'initialisation
Les premiers paramètres à déterminer sont la topologie, la forme et la taille (nombre de neurones) de la SOM. Ces paramètres ont chacun leur influence propre mais qui ne sera pas abordée dans cet article. Le point important lors de l'initialisation d'une SOM est la construction des vecteurs de référence de chaque neurone. La valeur des éléments de chaque vecteur de référence doit être aléatoire, ceci dans le but de copier encore une fois ce qui se passe dans le cerveau humain : à la naissance, les neurones sont organisés un peu n'importe comment, ce n'est qu'avec l'apprentissage que des connexions se forment et s'organisent. Il est évident que le nombre d'éléments dans les vecteurs de référence doit être égal à celui des vecteurs de données.

Composition des vecteurs (données et donc référence)
Chaput préconisait des vecteurs binaires, mais il est possible d'utiliser des valeurs entières directement ce qui a pour effet direct de réduire considérablement la taille des vecteurs et donc le temps de traitement par l'algorithme

(pour représenter 112 en binaire il faut 8 digits alors qu'il n'en faut que 1 en utilisant la valeur entière directement). Malgré cela, afin de pouvoir utiliser les vecteurs correctement dans l'algorithme, il faut que la valeur de chaque élément soit impérativement comprise entre 0 et 1, représentant ainsi la valeur relative et pourront alors être comparées entre elles (comme reporté en fig.4). Utiliser des vecteurs non-binaires permet également de traiter des valeurs négatives beaucoup plus facilement qu'avec une composition binaire, et donc l'utilisation de capteurs pouvant donner des valeurs négatives tels que les accéléromètres et les potentiomètres devient possible.

3.3 Entrainement et procédé de mise à jour d'une SOM

Entrainement

Comme expliqué dans les notions préliminaires, le processus d'entrainement implique la détermination d'un neurone vainqueur pour chaque donnée traitée par la SOM. Pour cela la donnée doit être comparée à chaque vecteur de référence afin de déterminer celui qui est le plus proche de la donnée. Le calcul de la proximité entre le vecteur de référence et le vecteur de données se fait en utilisant la Distance de Hamming légèrement modifiée (celle-ci étant à la base faite pour comparer des vecteurs binaires). La figure suivante explique comment comparer un vecteur de référence à un vecteur de données. A noter que dans cet exemple dans un souci de facilité d'explication, le vecteur de référence est binaire mais il ne doit pas l'être en réalité.

Input 1:	101	9	0	1	1	50
Max value:	360	15	1	1	1	60
Input / max	0.28	0.6	0	1	1	0.83
Reference:	1	1	0	0	1	1

Distance: 0.72 + 0.4 + 0 + 1 + 0 + 0.17 = 2.29

Input 2:	3	15	0	0	1	58
Max value:	360	15	1	1	1	60
Input / max	0.0083	1	0	0	1	0.97
Reference:	1	1	0	0	1	1

Distance: 0.9917 + 0 + 0 + 0 + 0 + 0.03 = 1.0217

Figure 4 : Distance de Hamming

On voit bien dans la figure 4 que la fonction de distance est adaptée : le vecteur de donnée 2 étant plus proche du vecteur de référence que le vecteur de données 1, la distance est plus petite.

Il est important que tous les éléments d'un vecteur soient compris dans le même intervalle de valeur (ici [0,1]). Si tel n'est pas le cas, les éléments dont la valeur est élevée faussent le calcul de proximité, comme l'illustre la figure 5 :

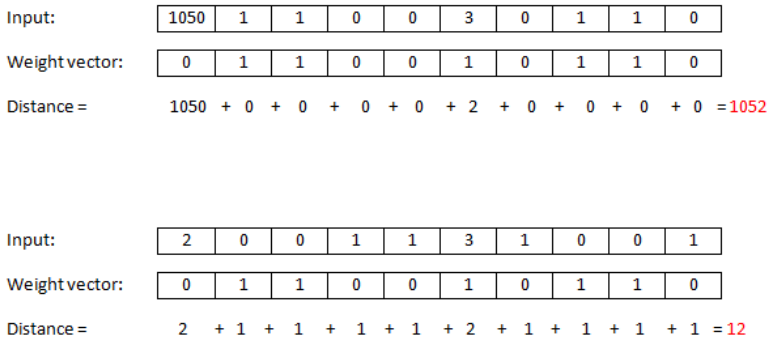


Figure 5: Importance d'un même intervalle de valeur

Mise à jour des vecteurs de référence

Ce procédé est la base de tout apprentissage non supervisé. On distingue deux procédés différents de mise à jour :

- Séquentiel
- Batch

Le procédé séquentiel réalise une mise à jour des vecteurs de référence à chaque fois qu'un vecteur de données est traité. Le procédé Batch, lui, fait la mise à jour des vecteurs de référence une fois que tous les vecteurs de données ont été traités et réalise plusieurs itérations complètes de ce procédé. Il est évident que le procédé Batch colle moins à la réalité. En effet, les neurones de notre cerveau ne se spécialisent pas une fois que nous avons fait tous les mouvements possibles sinon notre intelligence n'évoluerait jamais. Malgré cela, ce procédé a été utilisé dans ce travail car il accélère sensiblement l'algorithme et n'est pas contraignant à l'entrainement d'un robot. En effet, pour entrainer un robot à réaliser des actions, il est préférable de le soumettre directement à toutes les actions qu'il est capable de faire plutôt que de les lui proposer une par une.

La formule de mise à jour en procédé séquentiel est la suivante ;

$$m_i(t+1) = x(t) * \alpha(t) + m_i(t) * [1 - \alpha(t)]$$

Formule 2 : Mise à jour en procédé séquentiel

i = numéro du neurone
m = valeur d'un vecteur de référence
x = valeur d'un vecteur de données
t = temps discret
α = taux d'apprentissage

Le taux d'apprentissage est un facteur qui a toute son importance. Il permet de donner un poids plus ou moins important au vecteur de référence ou au vecteur de données. α a une valeur décroissante avec le temps ce qui permet que les premiers vecteurs traités par une SOM aient une influence plus importante lors de la mise à jour, et à l'inverse les vecteurs de données à la fin de l'apprentissage auront une influence moindre sur la valeur des vecteurs de référence après la mise à jour de ceux-ci.

La formule de mise à jour en procédé batch est la suivante :

$$m_i(t+1) = x_{i\ average}(t)$$

Formule 2 : Mise à jour en procédé batch

i = numéro du neurone
m = valeur d'un vecteur de référence
$x_{i\ average}$ = valeur moyenne des éléments « i » des vecteurs de données
t = itération

La valeur d'un vecteur de référence après la mise à jour sera donc égale à la valeur moyenne des vecteurs de données qui ont sélectionné ce neurone comme le vainqueur. A l'itération suivante, ce ne seront pas les mêmes vecteurs de données qui retrouveront ce même neurone comme vainqueur puisque la valeur du vecteur de référence aura changé. Une stabilisation est observée à partir d'un certain nombre d'itérations (cette valeur dépend de la taille de la SOM et des vecteurs).

A noter qu'il existe également une « option » supplémentaire au procédé de mise à jour : la fonction de voisinage. Celle-ci permet également de mettre à jour les neurones voisins du vainqueur.

Fin du processus d'apprentissage

Chaput dit que toutes les SOMs d'un niveau doivent être stables avant de pouvoir entrainer un niveau supérieur. Il est donc nécessaire de définir un critère de stabilité pour les SOMs. Par logique, la stabilité est atteinte lorsque la valeur des vecteurs de référence ne change plus de manière consistante. Il faut donc introduire un taux de stabilité. Celui-ci peut être simplement défini par la formule suivante :

$$taux = \sum_{n=1}^{nbr\ de\ SOMs} \frac{vect.référence(n)(t) - vect.référence(n)(t-1)}{nbr\ de\ neurone\ dans\ une\ SOM}$$

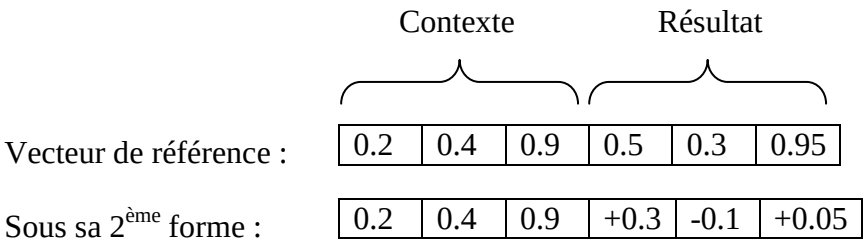
Formule 3 : Taux de stabilité

Où t = numéro de l'itération

3.4 Récolte et tri des schémas

Le principe est ici assez simple, il s'agit simplement de faire le tri parmi tous les vecteurs de référence une fois le processus d'entrainement terminé (lorsque la stabilité est atteinte). Parmi tous ces vecteurs, certains sont « intéressants » et d'autres non. Ceux qui sont intéressants sont ceux dans lesquels un changement significatif de l'état d'au moins un élément est observé. Une façon simple d'observer ces changements d'état est d'utiliser les vecteurs de référence sous leur seconde forme. Pour réaliser cela, il suffit de soustraire la partie contexte de la partie résultat. De cette manière, si un capteur est passé d'un état « OFF » à un état « ON » le résultat sera positif et, de la même manière, le résultat sera négatif dans le cas inverse. Il reste à déterminer si ce changement d'état est suffisamment significatif.

Pour cela une fonction de seuil est utilisée ; si le changement produit est au dessus de ce seuil le changement est significatif et le vecteur de référence devient un schéma. Dans le cas contraire il n'est pas utilisé.



Dans cet exemple on peut voir les 2 différents cas envisageables :

- Si le seuil est égal à 0.1, le vecteur de référence devient un schéma parce que ses éléments n°1 et n°3 du résultat sont >= 0.1.
- Si le seuil est égal à 0.4, il ne le devient pas parce qu'aucun de ses éléments du résultat n'est supérieur au seuil et donc un changement significatif ne s'est pas produit sur un des capteurs du robot.

Une fois le principe appliqué à tous les schémas, il reste à éliminer les doublons afin d'alléger la mémoire du programme. Le robot ne garde alors en mémoire qu'un seul exemplaire de chaque schéma, et il peut les utiliser plus tard afin de réaliser des mouvements en partant d'une situation pour arriver à une autre situation. Il est à noter qu'à chaque schéma correspond une action bien précise.

3.5 Insertion « d'éléments synthétiques » dans les niveaux supérieurs

Cette partie est sans conteste la plus importante ! Il s'agit ici de créer des interconnexions entre un niveau et son niveau directement supérieur. Les interconnexions sont créées en intégrant des informations apprises à un niveau bas dans les vecteurs de données d'un niveau haut. Pour cela il est nécessaire de transformer chaque schéma appris à chaque niveau en élément synthétique. Ceux-ci seront ensuite utilisés par le niveau directement supérieur.

Comme expliqué au point 2.2, un élément synthétique représente une situation dans laquelle le robot se trouve. Un élément synthétique donne une information sur la possibilité d'un mouvement ; si l'élément est « ON » (c'est à dire a la valeur « 1 »), le schéma correspondant à l'élément synthétique peut être réalisé et il ne le peut si l'élément est « OFF » (valeur « 0 »).

Ce principe n'étant pas des plus simples à comprendre au premier abord, une description plus détaillée de cette étape s'impose.

Le micro-monde

Afin d'illustrer le principe des éléments synthétiques, un micro-monde (fig. 6) a été défini. Ce micro-monde est une carte 5x5 dont les 25 coordonnées représentent les différentes positions que le robot peut prendre. Les actions que le robot peut réaliser sont des déplacements et sont au nombre de 4 : vers le haut, vers le bas, vers la gauche et vers la droite. Un objectif

doit également être défini afin de vérifier que le robot est bien capable d'atteindre un objectif et de sélectionner les schémas menant à ce même objectif. L'objectif à atteindre pour le robot est le point central de coordonnées (2,2).

Note : le robot ne peut se déplacer que d'un point vers un point directement voisin.

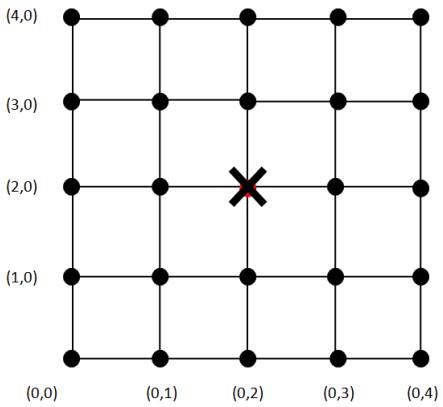
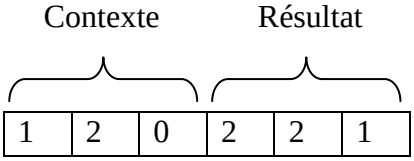


Figure 6: le micro-monde

Vecteurs de données

Les vecteurs de données sont donc composés de trois éléments pour le contexte et par conséquent de trois éléments pour le résultat. Les 2 premiers éléments représentent les coordonnées et le troisième est un élément binaire étant à « 1 » lorsque le robot se trouve sur l'objectif et à « 0 » lorsqu'il ne l'est pas.

Voici un exemple de vecteur de données : le robot est sur le point (1,2) et va sur (2,2), il se trouve alors sur l'objectif et le troisième élément prend la valeur 1.



Entrainement et récolte des schémas

La base de données pour l'entrainement du robot se constitue donc d'une multitude de vecteurs de ce type et reprend tous les mouvements possibles. Une base de données importante est préférable à une petite base de données, ainsi tous les déplacements sont représentés plusieurs fois ce qui est préférable dans un système d'apprentissage non supervisé tel que celui décrit dans cet article. Une base de données, (générée par programme ou par simulation en temps réel) de 10000 vecteurs est acceptable.

4 SOMs sont créées (car il y a 4 actions différentes) et sont chacune entrainées par les vecteurs correspondants. On peut dès lors statistiquement s'attendre à ce que chaque SOM soit entrainée par 2500 vecteurs. Une fois les SOMs entrainées les différents schémas sont récoltés. Avec une bonne fonction de distance couplée avec la fonction de voisinage adaptée et une taille de SOM optimisée, on devrait retrouver tous les différents déplacements que le robot peut faire au niveau des schémas.

Dans un souci de clarté d'explication, seuls les schémas amenant le robot à l'objectif seront retenus. Mais il est évident qui si les résultats sont valables pour un point, ils le sont également pour tout autre point.

Les schémas retenus sont donc les suivants (fig.7 et fig.8):

Context			Result		
2	1	0	2	2	1
1	2	0	2	2	1
2	3	0	2	2	1
3	2	0	2	2	1

Figure 7 :
Schémas retenus au 1er niveau

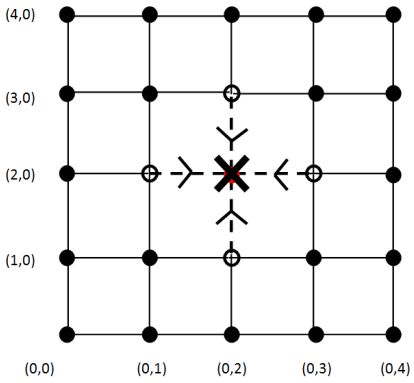


Figure 8 : Schémas retenus au 1er niveau

Insertion des éléments synthétiques

Le niveau le plus bas étant à présent entrainé, il s'agit d'entrainer le niveau supérieur. Il doit pouvoir utiliser les informations apprises par le niveau inférieur pour les coupler aux informations apprises à ce niveau. Pour cela, il est nécessaire que les vecteurs de données incluent les éléments synthétiques. Comme expliqué précédemment, les éléments synthétiques représen-

tent une situation, ou encore un schéma, et donnent une information sur la faisabilité de réalisation de ce schéma.
La partie contexte d'un vecteur de données peut être comparée à la partie contexte et à la partie résultat d'un schéma. Il en est de même pour la partie résultat d'un vecteur de données. 4 schémas ont été retenus au niveau 1, il y aura donc 2*4 éléments supplémentaires dans le contexte et le résultat d'un vecteur de données. Afin de déterminer les éléments synthétiques qui seront activés, il faut comparer les différentes parties, c'est-à-dire comparer :

- le contexte de la donnée avec le contexte du schéma,
- le contexte de la donnée avec le résultat du schéma,
- le résultat de la donnée avec le contexte du schéma,
- le résultat de la donnée avec le résultat du schéma.

Imaginons que le vecteur de données suivant doive être retraité pour y insérer les éléments synthétiques :

3	3	0	3	2	0

Les 2 parties du vecteur de données vont être comparées aux 2 parties des 4 différents schémas et lorsqu'une similarité est observée l'élément synthétique correspondant est activé (mis à « 1 »), comme l'illustre la figure 9 :

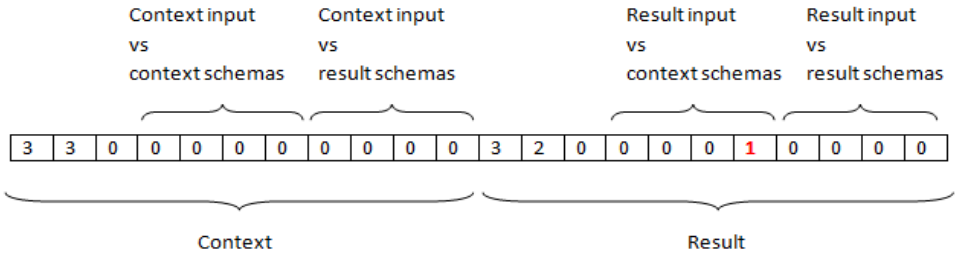


Figure 9 : Activation des éléments synthétiques

On peut voir dans ce schéma qu'un élément synthétique est activé, car le résultat du vecteur de données (3,3,0 - **3,2,0**) correspond au contexte du schéma numéro 4 (**3,2,0** – 2,2,1).
Ce qui veut dire dans ce cas que le robot reconnait que, partant de (3,3,0) pour se retrouver en (3,2,0) le place dans un état où il sait que se déplacer vers le bas l'amènera à l'objectif, car il a enregistré le schéma correspondant au niveau précédent.

Néanmoins, cela ne sera possible que si le schéma de la figure 9 est généré après entrainement des SOMs de niveau 2, ce qui n'est pas forcément le cas car les vecteurs d'entrée d'une taille importante sont plus difficiles à traiter et donc à transformer en schémas. Afin d'alléger le programme, on peut utiliser un subterfuge qui n'altère pas la justesse du processus (du moins dans ce cas). On peut entrainer les SOMs de niveau 2 comme au niveau 1, avec des vecteurs de données simplement constitués des coordonnées et de l'élément objectif atteint ou pas, générer des schémas et inclure les éléments synthétiques directement dans les schémas. Cela donne de meilleurs résultats si on n'utilise pas une fonction de distance optimisée.

Apprentissage de mouvements complexes

Au fur et à mesure des niveaux d'intelligence, le robot apprend des mouvements de plus en plus complexes en retrouvant des informations à travers les éléments synthétiques.

Les schémas suivants (fig.10 et fig.11) illustrent la façon dont le robot apprend, dans le cas du micro-monde, tous les déplacements qu'il peut faire pour atteindre le point central et ce, quel que soit sa position de départ.

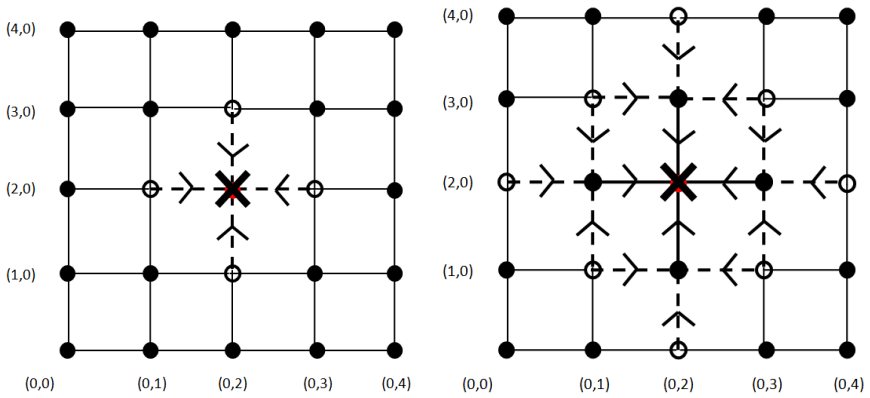


Figure 10 : Schémas de niveau 1 et 2

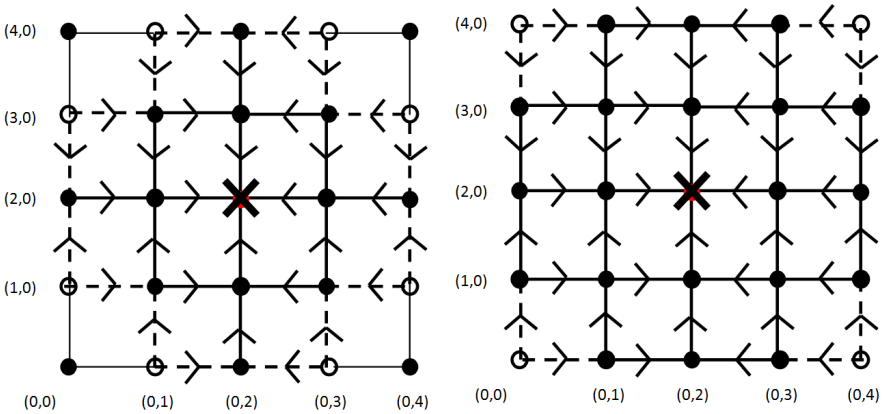


Figure 11 : Schémas de niveau 3 et 4

Le robot a donc appris à atteindre le point central depuis n'importe quel autre point en 4 niveaux. Il s'agit à présent de discuter les résultats et la façon de les faire utiliser par le robot.

4. Discussion des résultats

4.1 Utilisation des schémas par le robot

La discussion porte surtout sur comment interpréter et utiliser les schémas générés aux différents niveaux.
La première étape consiste à déterminer (dans le cas du micro monde) quel chemin le robot doit prendre. Deux situations peuvent être envisagées :

- le robot se déplace de point en point en suivant les lignes reliant les points ;
- le robot se déplace entre sa position et son objectif par le chemin le plus court, comme par exemple en diagonale.

Dans le premier cas, le robot doit entrer dans un processus décisionnel plus important et plus lent que dans le deuxième. Il doit pouvoir utiliser les éléments synthétiques de tous ses niveaux d'intelligence. Pour cela, à chaque niveau, il doit retrouver à quel schéma du niveau inférieur correspond l'élément synthétique activé dans un schéma d'un niveau. Une fois retrouvé, il doit réaliser la même opération avec le niveau inférieur et ainsi de suite jusqu'au bout de l'arborescence des niveaux d'intelligence. Ce processus demande du temps et doit être convenablement programmé de façon optimale.

Dans le deuxième cas on peut imaginer que le robot se rende à son objectif par le chemin le plus court possible, dans ce cas en diagonale. Pour cela il doit simplement se référer à ses schémas de niveau haut et en utiliser le contexte et le résultat. Il sait alors quelle distance en X et en Y le sépare de l'objectif et peut calculer par Pythagore l'angle dans lequel il doit se déplacer et la longueur de déplacement. La figure 12 présente les deux cas envisageables :

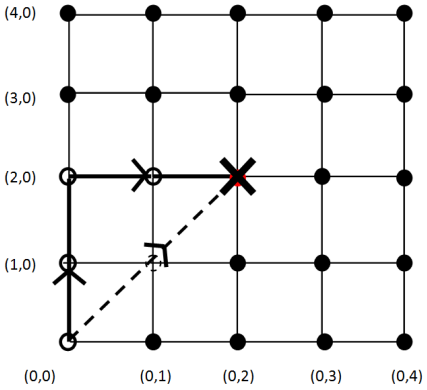


Figure 12 : Interprétation des résultats

4.2 Condition d'arrêt de l'algorithme

Une discussion sur la condition d'arrêt de l'algorithme mérite d'avoir lieu. Trois différentes conditions d'arrêt peuvent être imaginées :

- L'algorithme doit s'arrêter quand il a appris comment rejoindre les 2 points les plus éloignés, c'est-à-dire, aller de (0,0) à (4,4), ce qui requiert 8 niveaux d'intelligence.
- L'algorithme doit s'arrêter quand il a généré tous les différents schémas possibles. Ce qui équivaut à dire que le chemin le plus long possible sans retour à une précédente position est appris par le robot, ce qui requiert 24 niveaux d'intelligence (fig.13).

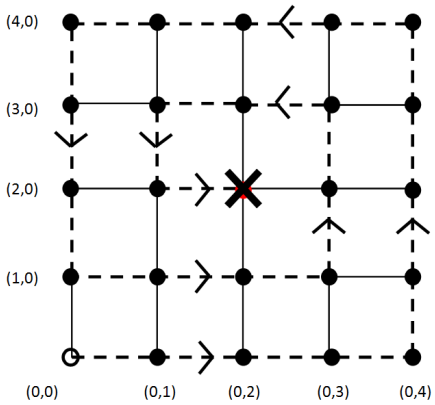


Figure 13 : Condition d'arrêt n°2

- La troisième condition (fig.14), qui est la plus obsolète, serait que l'algorithme s'arrête après un nombre déterminé de niveaux d'intelligence. Bien évidement ceci est absurde dans le sens que nous humains ne fonctionnons pas de cette manière. Cette condition d'arrêt permettrait néanmoins que le robot apprenne à faire des boucles, c'est-à-dire revenir sur un ou des même(s) point(s), une ou plusieurs fois, avant de rejoindre son objectif.

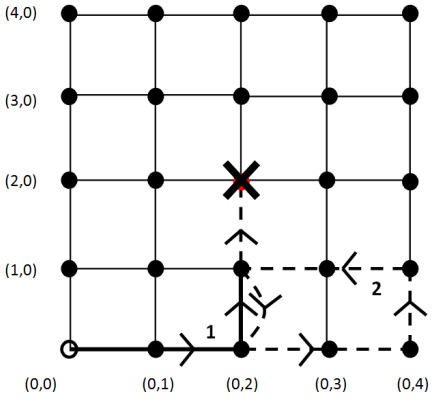


Figure 14 : 3ème condition d'arrêt

Le choix de la bonne condition d'arrêt dépend de ce que le robot est supposé apprendre. Si le robot est censé apprendre comment rejoindre rapidement l'objectif, la première condition est la plus adaptée. Les deux autres sont plus adaptées pour un modèle dans lequel le robot pourrait apprendre absolument tous les mouvements qu'il peut faire, comme par exemple faire 100

fois le tour du micro-monde avant de rejoindre le point central. Néanmoins ce genre de mouvement peut être réalisé en utilisant la première condition d'arrêt en manipulant correctement les schémas des différents niveaux.
La première solution est certainement meilleure dans la plupart des cas. En considérant que les algorithmes présentent généralement une certaine caractéristique d'optimisation, elle est plus adaptée. Mais encore une fois, cela dépend de l'application que l'on veut donner au robot.

4.3 Changements dans l'environnement

La question fondamentale à se poser, et qui est au centre de ce sujet, est de se demander ce qu'il se passerait si l'environnement changeait. Comment le robot réagirait si le micro-monde venait subitement à s'agrandir ? Comment s'adapterait-il si soudainement un mur apparaissait dans le micro-monde ? Comment pourrait-il l'éviter et réapprendre de nouveaux déplacements en tenant compte de ce nouvel élément ?

Ces différentes questions, aussi simples soient-elles, nécessitent une réflexion approfondie. C'est ici la question de l'adaptation qui est en jeu, et pouvoir y répondre est crucial si on veut créer un robot doté d'une intelligence proche de l'homme. Il faut savoir qu'une des meilleures définitions de l'intelligence est simplement la capacité d'adaptation.
Une partie de la réponse se trouve dans les explications de Chaput. Afin de pouvoir s'adapter, le robot doit pouvoir créer de nouveaux neurones dans ses différentes SOMs. Ces nouveaux neurones pourraient alors être entrainés à traiter les nouvelles informations, les neurones voisins pourraient également en ressentir l'effet si une fonction de voisinage est utilisée. Chaput suggère que ces extensions du nombre de neurones ne sont pas obligatoires à tous les niveaux, mais uniquement à un seul niveau. C'est ce qu'il appelle le « gracefully fallback ». En reprenant l'exemple du dactylographe, s'il passe d'un clavier AZERTY à un clavier QUERTY il n'a pas besoin de réapprendre les associations de lettres pour écrire des mots, il doit simplement réapprendre au niveau le plus bas la position des lettres.

5. Conclusion et suite du travail

Cet article résume un travail de fin d'études réalisé à l'Université d'Aberdeen dans le cadre d'un Erasmus de 5 mois. Ce projet, basé sur 10 ans, a été demandé par une compagnie pétrolière d'Aberdeen, et le travail décrit dans cet article est le tout début du développement de ce projet ambitieux. Une bonne base pour un futur développement a été posée : savoir qu'un algorithme d'apprentissage simple a été développé et donne des résultats encourageants pour la suite. Le point sans doute le plus important développé ici est l'utilisation correcte des éléments synthétiques à travers les différents niveaux, et qui symbolisent les interconnexions nerveuses d'un cerveau biologique. Certains principes relatifs à l'intelligence humaine sont respectés :

- Le robot ne sait rien avant de démarrer son apprentissage, aucune fonction n'est préétablie dans son système.
- Il apprend de lui-même.
- Il associe des mouvements simples afin d'en faire des mouvements plus complexes, tout en utilisant différents niveaux d'intelligence.

Le processus de Chaput a été remis en question et a été amélioré, notamment en permettant d'utiliser directement des valeurs réelles et non plus de les utiliser sous leur forme binaire. Les résultats ont été sensiblement meilleurs : l'algorithme est beaucoup plus rapide, donne des résultats beaucoup plus corrects, et l'utilisation de capteurs renvoyant des valeurs négatives est largement facilitée.
Néanmoins et comme évoqué ci-dessus, ce travail est une base. Un développement énorme doit suivre. Le robot doit pouvoir se doter d'une véritable faculté d'adaptation, lui permettant de réagir de la meilleure façon qui soit face à n'importe quelle situation. L'objectif final est qu'un robot entrainé dans un milieu, doit pouvoir être utilisé dans un milieu complètement différent sans pour autant être désorienté.
Les pistes à suivre pour un développement futur sont :

- Améliorer la fonction de distance afin de pouvoir inclure les éléments synthétiques directement dans les vecteurs d'entrée et que ceux-ci puissent être transformés en schémas après entrainement tout en gardant un sens.
- Implémenter une fonction de voisinage.
- Implémenter une fonction de seuil qui répond correctement à la dispersion de valeurs des vecteurs d'entrée.

- Améliorer l'algorithme afin que de nouveaux neurones puissent être créés automatiquement et au bon niveau lorsque le robot se retrouve face à une situation qu'il ne connait pas.
- Inclure des notions d'apprentissage par récompense dans l'algorithme.

6. Sources

[1] CHAPUT Harold Henry. *The Constructivist Learning Architecture: A Model of Cognitive Development for Robust Autonomous Robots*, University of Austin, Texas.

[2] LEMAIRE Vincent, *Cartes auto-organisatrices pour l'analyse de données*, Université de Lyon, France.

[3] Somtoolbox description,
Adresse URL : http://www.cis.hut.fi/somtoolbox/

[4] Unsupervised learning,
Adresse URL : http://en.wikipedia.org/wiki/Unsupervised_learning.