UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Apprentissage Machine pour un système multi-robot autonome

ARASH SADEGH
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE AUX SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
6 AVRIL 2008

Your file Votre référence
ISBN: 978-0-494-41578-8
Our file Notre référence
ISBN: 978-0-494-41578-8

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

Apprentissage Machine pour un système multi-robot autonome

Présenté par: SADEGH Arash
en vue de l'obtention du diplôme de: Maîtrise des sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. GOURDEAU RICHARD, Ph.D., membre et directeur de recherche
M. HURTEAU RICHARD, Ph.D., Président
M. BRAULT JEAN-JULES, Ph.D., membre

REMERCIEMENTS

Tout d'abord je tiens à remercier mon directeur de recherche, M. Richard Gourdeau, pour son soutien, ses encouragements, son support et la disponibilité dont il a fait preuve tout le long de ma maîtrise.

J'exprime également ma reconnaissance auprès, de tous les membres de l'équipe de robotfoot ainsi que le personnel administratif du département de génie électrique pour leur support de chaque jour.

J'exprime également ma reconnaissance pour le support financier accordé par le Fond de Recherche sur la Nature et Technologie du Québec.

Toute ma gratitude envers ma famille qui m'a supporté sans relâche dans mon choix de carrière et m'a encouragé à persévérer.

Et finalement, je tiens à remercier globalement les professeurs de l'École Polytechnique pour leur ouverture d'esprit et pour leur volonté à accepter de discuter, d'analyser et de commenter avec moi les différents aspects de la technologie.

RÉSUMÉ

Ce projet de maîtrise a permis de développer un système d'apprentissage machine pour un système multi-robots. Le domaine d'utilisation de ce système étant celui d'une équipe de véhicules autonomes effectuant un travail collaboratif avec synchronisation. Les robots sont des agents pouvant, périodiquement, communiquer entre eux leurs états et leurs décisions, soit sans aucune restriction de communication autre que la période d'exécution (pendant que l'équipe est hors-ligne par exemple lors des pauses) ou avec un certain compromis entre la performance et la communication (pendant que l'équipe est enligne par exemple lors d'une phase de jeu).

Les robots joueurs de soccer conçus par la société étudiante *robofoot* sont utilisés dans le cadre des travaux de recherche. Il s'agit d'une équipe composé de 6 robots joueur de soccer. L'approche utilisée divise le problème d'apprentissage en sous-parties afin de tenter de développer les techniques d'apprentissage à partir des comportements de bas niveau et les combiner pour progresser vers les comportements de haut niveau. En utilisant cette approche d'apprentissage multi niveaux (hiérarchisé), différents algorithmes d'apprentissage peuvent être utilisés pour chaque niveau. Le résultat d'apprentissage de chaque niveau est donc utilisé par un niveau supérieur.

Les contextes d'apprentissages étant très vastes et différents, un système d'apprentissage d'entrée-sortie générique et hiérarchisé est mis en place permettant d'apprendre des comportements très simples (par exemple l'interception du ballon ou le tir au but) qui sont ensuite utilisés par des comportements en équipe (effectuer une passe) et finalement dans des stratégies de jeu.

Également, les différentes méthodes d'apprentissage sont analysées et comparées à chaque niveau. Plusieurs algorithmes (méthodes) d'apprentissage ont été étudiés et les méthodes d'apprentissage par machine à vecteurs de support (*Support Vector*

Machines), par perceptron multicouche (*Multi Layer Perceptron*) et une méthode d'apprentissage basé-mémoire ont été choisies pour faire l'apprentissage des robots joueurs de soccer.

ABSTRACT

The main goal of this master's thesis is to develop a system which allows machine learning for a multi-robot system. The main domain of application of this system is one which constitutes of a group of autonomous vehicles working synchronously together. The robots are agents capable of communicating their state and decisions amongst each other periodically without any restrictions (during timeouts) or by taking into account certain compromise between their performance and their communication. These robots have been developed and maintained by the *robofoot* student community in *École de Polytechnique de Montréal.*

The problem of learning has been separated into simple components in order to allow the learning process to be started from lower level behaviors and actions and combine those in order to allow the learning of higher level decisions. Using this multi-level approach, not only different algorithms and methods of learning can be compared with each other at each level, but also we have the possibility to use different algorithms at different levels.

Given the vast context of the learning of such a system, a generic hierarchical input-output learning system has been developed which allows learning simple behaviors such as catching or shooting a ball, and uses them to learn a collective behavior such as a pass. Finally the team strategy as the ultimate collective decision can be learned. Also, in order to have a more efficient learning process, a forgetting factor has been studied.

The different methods and algorithms which have been used in this project are the *Support Vector Machines* algorithm (SVM), a variant of neural networks called the *Multi Layer Perceptron* algorithm and also a memory-based learning method.

Table des Matières

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ANNEXES

LISTE DES SYMBOLES, SIGLES ET ABRÉVIATIONS

a	Action à effectuer utilisant une apprentissage par renforcement
b	Constante d'une équation linéaire
bb	botter ballon (une fonction dans SUI)
AE	Entropie moyenne (Average Entropy)
AHC	Critique heuristique adaptive (Adaptive Heuristic Critic)
C	Facteur de compromis dans un SVM
d	Diamètre (m)
e	Erreur entre la valeur désirée et la valeur obtenue
ε	Epsilon (valeur d'erreur permis dans un SVM)
ξ_l	Marge d'erreur dans un SVM
H	Entropie
HDM	Machine hiérarchisée décisionnelle (Hiearchical decisional machine)
K(.,.)	Fonction de Kernel
v	Vitesse (m/min)
MLP	Une variante de réseaux neurones intitulé Multi Layer Perceptron
MSE	Erreur au carrée moyenne (Mean Squared Error)
p	Proportion positive d'exemples dans l'ensemble des données pour le calcul d'entropie
PID	Proportionnel-intégral-dérivé
Q	Fonction d'évaluation ou de coût
r	Le signal de renforcement associé à une action lors d'une apprentissage RL
R_L	Risque empirique
RL	Apprentissage par renforcement (Reinforcement Learning)
S	État interne utilisé pour l'apprentissage par renforcement
SVM	Machine à vecteur de support (Support Vector Machine)

SUI	Interface usager de serveur (Server User Interface)
X	Vecteur de paramètres d'entrée pour un algorithme d'apprentissage
Y	Vecteur de paramètres de sortie pour un algorithme d'apprentissage
v	Critique utilisé pour un apprentissage AHC
γ	Facteur de régularisation pour un MLP
λ	Facteur d'oublie dans un apprentissage machine
φ	Fonction de transfert pour un apprentissage par MLP
η	Coefficient du taux d'apprentissage pour un apprentissage par MLP
σ	Paramètre d'ajustement de Kernel gaussien

Introduction

Ce document traite les différents aspects des travaux réalisés dans le cadre du projet de maîtrise portant sur le développement d'un mécanisme d'apprentissage pour un système multi-robots à l'École Polytechnique de Montréal.

Le texte qui suit présente une étude sur l'apprentissage machine, le contexte de développement du projet, les différents éléments développés, les résultats obtenus ainsi que leur analyse et une discussion sur ces résultats.

0.1. Motivations

Pourquoi l'homme cherche toujours à apprendre plus? Est-ce que ceci fait parti de sa nature d'évoluer ou c'est tout simplement c'est un besoin qu'il a besoin de satisfaire? Tout comme les êtres humains, les robots doivent aussi pouvoir évoluer et apprendre afin de pourvoir fonctionner de façon autonome.

Aujourd'hui, la science de la robotique maîtrise assez bien l'aspect du contrôle des robots, et l'on peut facilement imiter des mouvements de l'être humain mais l'aspect intelligence artificielle et de l'apprentissage machine est encore très peu développé pour pouvoir être comparé à celui d'un être humain.

Aussi, le fait de pouvoir utiliser un groupe de robots permet d'étudier des comportements collectifs et de pouvoir non seulement utiliser le résultat obtenu pour des projets semblables tel que des missions de recherche et sauvetage, mais aussi d'introduire et étudier certains avantages qu'un système multi-robots possède par rapport à un seul robot ; une meilleure tolérance aux fautes grâce au fait qu'ils sont

interchangeables ou une performance améliorée grâce à la combinaison des ressources.

0.2. Plateforme d'essai : soccer robotisé

La plateforme utilisée dans le cadre de ce projet est celle du soccer robotisé. Tout comme la majorité des systèmes multi-robots, le développement d'une équipe autonome de robots joueurs de soccer demande la mise en œuvre d'un système complexe touchant plusieurs domaines d'ingénierie tel que l'apprentissage machine, l'informatique temps réel et contrôles de haut et de bas niveau.

Le soccer étant le sport le plus populaire au monde, plusieurs compétitions annuelles sont organisées à travers le monde pour le soccer robotisé. Ce qui permet de non seulement évaluer les performances de chaque équipe mais aussi de regrouper la communauté scientifique intéressée par les domaines en question. La plus importante compétition internationale de soccer robotisé est la *Robocup* qui permet aux équipes provenant de tous les coins du monde de s'affronter une fois par année.

Les principaux modules de ce système s'agissent :

- Une plateforme électromécanique,
- Un logiciel de contrôle temps-réel contenant le module de commande des actuateurs, le module de cognition et la module de perception.
- Une plateforme en simulation contenant entre autres un module qui simule les mouvements des robots et un logiciel qui simule la dynamique du ballon ainsi qu'un visualisateur 3D complétant ce simulateur.

Chapitre 1
Apprentissage machine : Méthodes à explorer

La robotique est un domaine en pleine croissance et à chaque jour, nous trouvons de plus en plus d'outils automatisés dans nos vies. Les outils automatisés que nous utilisons sont pour la plupart dédiés à une tâche spécifique mais une partie est dédiée à des tâches plus complexes qui nécessitent une coopération entre divers outils ou robots, par exemple les machines automatisées utilisées dans les usines de fabrication d'automobiles.

Ce projet porte sur le développement d'un système d'apprentissage machine pour un système multi-robots. Après avoir mis en contexte le projet, et y avoir expliqué la nécessité de l'apprentissage et les méthodes à explorer pour cette fin, différentes méthodes d'apprentissage serons traitées dans ce chapitre.

1.1. Contexte

Le domaine d'utilisation de ce système étant celui d'une équipe de véhicules autonomes effectuant un travail synchronisé. Dans le cadre de ce projet, les robots conçus par la société étudiante *robofoot* ont été utilisés. Leur mission est de gagner un match de soccer contre des équipes adverses qui sont elles aussi des robots joueurs de soccer. Une description complète de la plateforme utilisée est fournie dans le deuxième chapitre de ce document.

1.2. Besoin d'apprentissage

Tout comme les êtres humains, les robots doivent apprendre afin de s'adapter rapidement aux variations de l'environnement. Les robots joueurs de soccer possèdent chacun des comportements de base prédéfinis. C'est-à-dire que chaque robot, dépendamment de son rôle dans l'équipe et la situation de jeu, choisit un comportement préétabli. Mais ces comportements utilisés par les robots ne seront pas optimaux si on y utilise des paramètres fixes. Par exemple, lors d'interception du ballon plusieurs paramètres inconnus vont influencer la manière d'intercepter le ballon tel que le niveau de gonflement du ballon, la friction entre le terrain et le ballon. Ces paramètres ou les paramètres qui en dépendent ne sont jamais entièrement mesurables et ne peuvent être appris que pendant un match.

1.3. Méthodes d'apprentissage

L'approche utilisée est celle de diviser le problème d'apprentissage en sous-parties et d'apprendre des comportements individuels et simples et les combiner pour apprendre des comportements en équipe plus complexes. Les sections suivantes traitent des différentes méthodes d'apprentissages utilisées dans le domaine d'apprentissage machine. Dans le cadre de ce projet, les différentes méthodes d'apprentissage choisies (pour des raisons expliquées à la fin de ce chapitre) sont des méthodes basées sur des modèles mathématiques tels que la méthode de l'apprentissage par machine à vecteur de support, celle par réseau de neurones ainsi qu'une méthode d'apprentissage par expérience.

1.4. Comportement adaptatif et l'apprentissage

Au début, toute l'intelligence d'un agent provenait de la représentation du monde développée par son concepteur. Cette approche n'était certainement pas la meilleure, tant pour l'agent que pour le concepteur puisque celui-ci n'a souvent qu'une

connaissance incomplète de l'environnement dans lequel l'agent évolue. L'apprentissage peut être alors considéré comme une voie possible pour que l'agent acquière les informations nécessaires à un bon fonctionnement. Aujourd'hui, l'apprentissage est considéré comme une partie essentielle d'un système intelligent

Plusieurs définitions ont été données pour l'apprentissage ou un comportement adaptatif telles que « Modification de comportement basé sur l'expérience » (Webster 1984) ou « tout changement dans un système qui permet une amélioration de performance lors de la répétition de la même tâche ou une autre tâche révélant de la même population dans un environnement donné » (Simon 1983). D'après Arkin (Arkin 1998), l'apprentissage produit des changements à l'intérieur d'un agent autonome.

L'induction est la forme la plus commune d'apprentissage. Ceci revient à prédire le comportement futur en se basant sur les résultats du passé par exemple en sachant que les dernières centaines de personnes qui ont été à l'hôtel x, ont été satisfaites du service, on peut prédire (apprendre) que le service sera satisfaisant à cet hôtel.

Vue de façon technique, T. Lozano-Pérez du MIT distingue trois formes d'apprentissage (Lozano-Pérez 2005) :

1. Apprentissage supervisé : ayant obtenu une série de paires d'entrée/sorties, il s'agit de trouver une loi ou une fonction qui prédit de façon appropriée le lien entre les sorties et les entrées.
2. Groupement ou *Clustering* : ayant recueilli une série d'exemples, il s'agit de les rassembler dans des groupes naturels en se basant sur leurs caractéristiques (par exemple, regrouper des personnes selon leur groupe sanguin).
3. Apprentissage par renforcement : un agent effectue des actions en observant le monde autour de lui et il est récompensé ou puni dépendamment du fait que

son action a été appropriée ou non. Cette méthode diffère de l'apprentissage supervisé puisque personne ne précise la bonne action au robot et il doit trouver la bonne action en subissant les conséquences de chaque action. Cette forme d'apprentissage sera étudiée de façon plus détaillée dans la section suivante.

Peu importe l'approche utilisée, la figure 1.1 montre le modèle utilisé pour l'apprentissage machine, où l'environnement fourni à l'élément d'apprentissage certaines informations dont celui-ci utilise afin d'améliorer sa base de données des connaissances et finalement l'élément de performance utilise cette base de données pour performer une tâche.

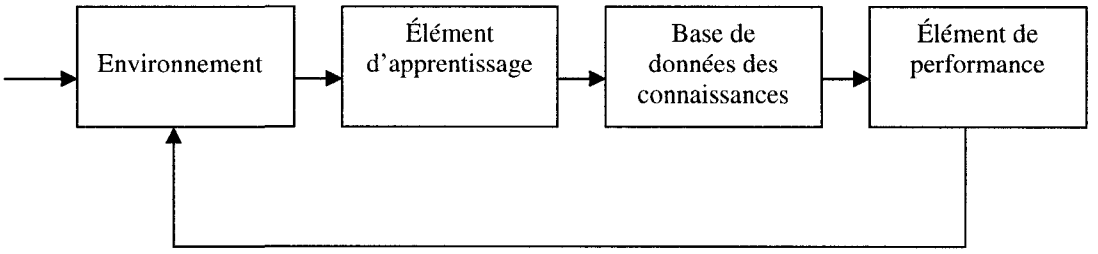


Figure 1.1 : Un model simple de l'apprentissage machine

Les recherches en intelligence artificielle ont conduit à plusieurs mécanismes qui permettent à un système robotique d'apprendre par lui-même. La section suivante présente les approches les plus intéressantes

1.5. Théories de l'apprentissage par prédiction

Une méthode d'apprentissage par prédiction, comme celle de machine à vecteur de support ou le SVM, nécessite d'avoir un certain nombre d'exemples ou de **données d'entraînement** à priori afin de pouvoir bâtir un **modèle** (trouver une relation **f**) qui

met en correspondance les entrées observées x_1, x_2, $x_3 ... \in R$ avec la sortie $y \in Y$ en utilisant une fonction de coût $\boldsymbol{Q}$ qui mesure la performance de notre modèle. Donc l'apprentissage revient à minimiser cette fonction de coût en utilisant les données d'entraînement.

Soit z un vecteur qui contient [x_1 x_2 .... x_n y], on peut alors formuler l'apprentissage comme un moyen de minimiser l'erreur E suivante (le *risque attendu*) en utilisant l'ensemble de donnée (entrées/sorties) :

$$R: f \in F \mapsto E(Q(z,f)) = \int_z Q(z,f)P(z)dz \tag{1.1}$$

Où la distribution P n'est pas connue. Alors, on peut essayer de minimiser ce que l'on appellera le *risque empirique*, soit R_L :

$$R_L:: f \in F \mapsto \frac{1}{L}\sum_{l=1}^{L} Q(z_l, f) \tag{1.2}$$

En fait c'est possible de démontrer (V. N. Vapnik et AY Chervonenkis) que la valeur de R_L tende vers R en augmentant le nombre de données d'entraînement L vers l'infini.

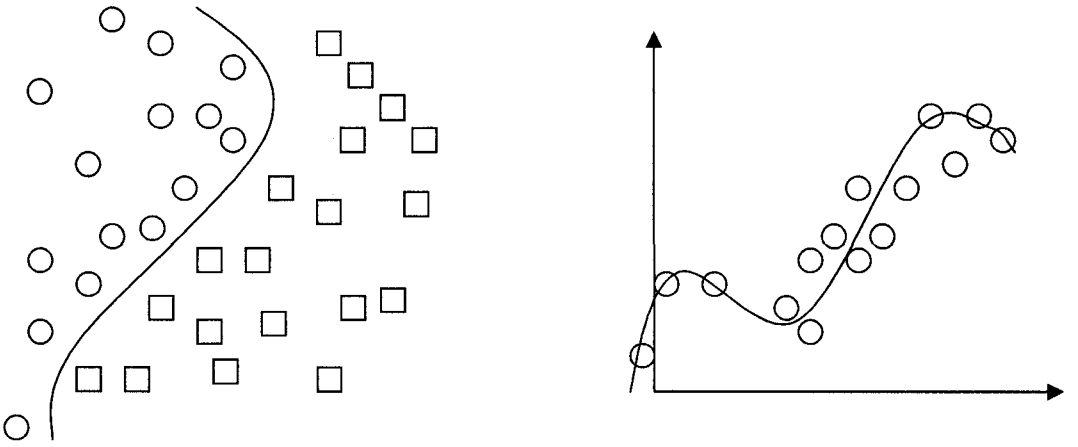
Il existe deux formes d'apprentissage par prédiction (voir la figure suivante):

- Par **classification** : dans ce cas la sortie y correspond à un ensemble fini Y (par exemple on peut classifier un être humaine comme étant un enfant, un adolescent ou une adulte), la fonction $\boldsymbol{Q}$ de coût est tout simplement dans ce cas

$$\boldsymbol{Q} = \begin{cases} 0 & \text{si } f(x) = y, \\ 1 & \text{sinon} \end{cases} \tag{1.3}$$

- Par **régression** : dans ce cas la sortie y fait parti d'un ensemble infini Y de nombre réel, et nous voulons trouver la fonction $f(x)$ qui donne la meilleure approximation de la bonne valeur de sortie y. On peut dans ce cas mesurer l'erreur entre la prédiction et la vraie valeur par une mesure statistique comme l'erreur au carrée moyenne (*MSE*). Dans ce cas la fonction de coût prend la forme suivante :

$$Q = || f(x) - y ||^2 \qquad (1.4)$$



(a) (b)

Figure 1.2 : Méthode d'apprentissage par classification (a) utilise dans ce cas un classificateur (une décision) qui sépare deux ensemble de données tandis que dans (b) on utilise la méthode de régression pour prédire la valeur de la sortie y en fonction d'entrée x (adapté de R. Collobert)

Dans le cadre des travaux de cette maîtrise, c'est évidemment l'apprentissage par régression qui est utilisée. Dans le cas d'apprentissage par régression nous voulons trouver un modèle :

$$x \rightarrow f_\Theta(x) \qquad (1.5)$$

de sorte que pour un exemple (x,y), la sortie y est prédite en fonction du vecteur d'entrée x c'est-à-dire que $y = f_\theta(x)$. Où θ représente les paramètres qui doivent être entraînés pour bâtir le modèle.

1.6. Réseau de neurones

Dans un système réel, le nombre de combinaisons situation/action peut être très élevé d'où la nécessité de conserver les informations dans un réseau de neurones. Un réseau de neurones formels est constitué d'un grand nombre de cellules de base interconnectées.

De nombreuses variantes sont définies selon le choix de la cellule élémentaire, de l'architecture du réseau et de la dynamique du réseau. L'architecture du réseau peut être sans rétroaction (c'est-à-dire que la sortie d'une cellule ne peut influencer son entrée) ou avec rétroaction totale ou partielle. Une cellule élémentaire peut manipuler des valeurs binaires (0,1 ou -1,1) ou réelles. Différentes fonctions peuvent être utilisées pour le calcul de la sortie. Le calcul de la sortie peut être déterministe ou probabiliste. La dynamique du réseau peut être synchrone (toutes les cellules calculent leurs sorties respectives simultanément) ou asynchrone.

Le but est de traverser tous les neurones pour arriver à un neurone de sortie. Pour un traverser un neurone il faut que la somme des valeurs des entrées x multiplié par des coefficients w_i soit plus grande qu'une valeur limite θ. Donc, la valeur de ce neurone (dans le cas binaire) sera 1 si ceci est vrai est 0 sinon.

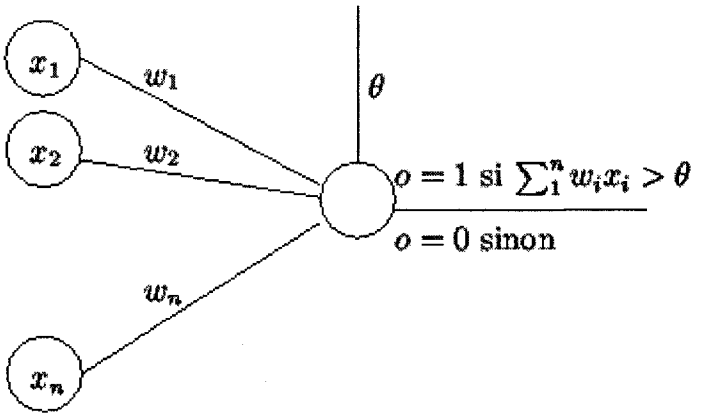


Figure 1.3 : Exemple d'un perceptron

Le but de la phase d'apprentissage est de déterminer les poids des connexions et le seuil des neurones. Cette phase dépend beaucoup de la structure du réseau. Normalement, l'apprentissage est *supervisé* c'est-à-dire qu'on teste le réseau dans des situations connues et on cherche à obtenir la sortie voulue. On effectue alors une modification des poids pour retrouver cette sortie imposée. Il existe aussi des réseaux à apprentissage *non-supervisé* ou capable de mémoriser. Dans ce cas, un raisonnement est fait par analogie avec ce qu'ils ont déjà effectué. Enfin certains réseaux associent ces deux types d'apprentissage.

1.6.1. Apprentissage par la méthode de perceptron multicouches (MLP)

On peut combiner plusieurs *Perceptrons* d'une manière non linéaire pour former un *Multi Layer Perceptron* (MLP) ce qui permet d'élargir les capacités de séparation du système d'apprentissage.

Un MLP est le résultat de la combinaison de plusieurs perceptrons et agit comme une fonction composée de plusieurs couches. Chaque couche contient plusieurs *Perceptrons* et leur sortie est transférée à la couche suivante en passant par une fonction de transfert qui est souvent la fonction sigmoïde suivante :

$$\varphi(x) = \frac{1}{1+\exp(-x)} \qquad (1.6)$$

Les couches intermédiaires sont intitulées les couches cachées et la dernière couche est la couche de sortie. Dans le cas de régression, la couche de sortie contient autant de *Perceptrons* que de dimension de notre vecteur de sortie *y*.

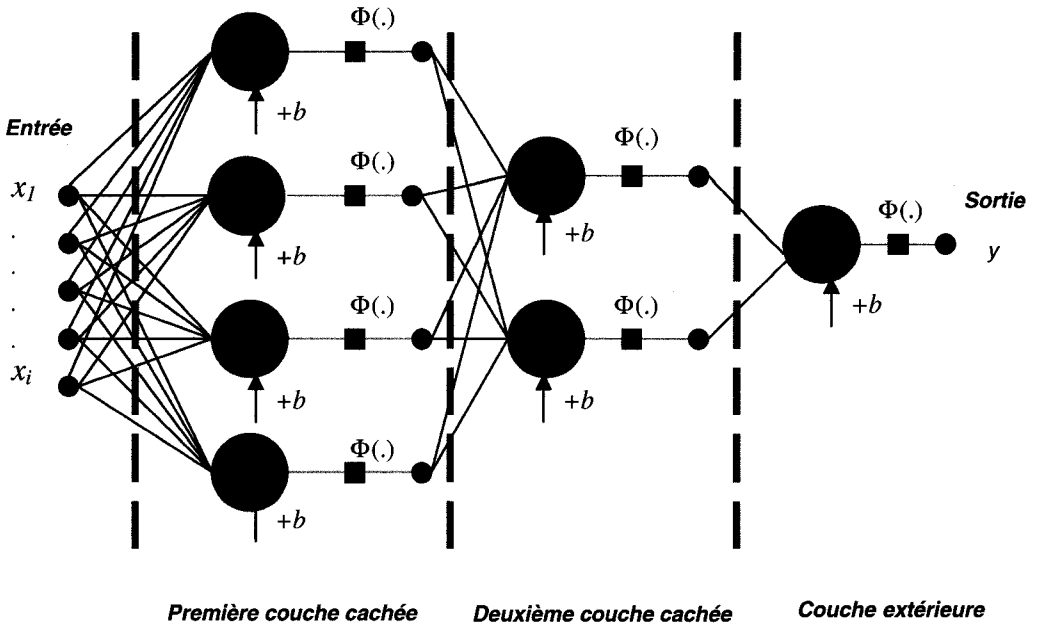


Figure 1.4 : Architecture d'un MLP

Dans le cadre de ce projet, un MLP avec une seule couche cachée sera étudiée parce que tel que démontré par Hornik et al.(1989), il existe un fonction de MLP f_θ qui permet d'approximer la valeur de la sortie y avec une précision connu étant donnée un nombre défini d'exemple d'entraînement.

La méthode utilisée généralement pour mettre à jour un coefficient est celui de rétro-propagation de l'erreur (puis qu'elle fait propager l'erreur de la sortie vers

l'intérieur).

Pour un neurone au niveau *j* dont la valeur de sortie à partir des nœuds interne *i* est définit par la formule suivante :

$$\begin{aligned} y_j(t) &= \varphi(v_j(t)) \qquad avec \\ v_j(t) &= \sum_{i=0}^{m} w_{ij}(t)\, y_i(t) + b \end{aligned} \tag{1.7}$$

Où φ est la fonction de transfert et w_{ij} est le poids qui connecte les nœuds *i* et *j*. Définissons l'erreur à la sortie d'un neurone *j* par la formule suivante :

$$e_j(t) = d_j(t) - y_j(t) \tag{1.8}$$

Où *y* la valeur de sortie et *d* est la valeur désirée de la sortie. Alors la valeur du poids entre les nœuds *i* et *j* est défini par la formule suivante :

$$w_{ij}(t+1) = w_{ij}(t+1) + \eta \delta_j y_i(t) \tag{1.9}$$

Où :

- y_i (t) est la sortie du nœud i,
- η est le coefficient du taux d'apprentissage (déterminé normalement par un processus d'essai et d'erreur),
- $\delta_j(t) = e_j(n)\varphi_j'(v_j(t))$ pour un nœud de sortie et

  $\delta_j(t) = \varphi_j'(v_j(t))\sum_i \delta_i(n) w_{ij}(n)$ pour des nœuds internes.

Il y a deux modes pour mettre à jours les coefficients *w* d'un réseau de neurone. Le mode séquentiel et le mode en batch. Le mode séquentiel ou on-line s'agit de mettre à

jour les poids après avoir l'introduit chaque donnée d'entraînement dans le système, tandis que le deuxième mode est basé sur le fait que les poids sont mis à jours utilisant tous les données d'entrainements.

L'entraînement de MLP est achevé utilisant un ensemble de données d'entraînement et en optimisant un critère. Le critère peut être celui de MSE (Mean Squared Error), soit :

$$Q((x,y),f_\theta)=\frac{1}{2}\|y-f_\theta(x)\|^2 \qquad (1.10)$$

Afin d'incorporer un facteur d'oubli dans ce critère, il vaut mieux remplacer le critère de l'équation 1.10 par la formule suivantes :

$$Q((x,y),f_\theta)=\sum_{s=1}^{t}\lambda^{t-s}\quad\|y(s)-f_\theta(x(s))\|^2 \qquad (1.11)$$

Ce qui nous permet de minimiser le risque empirique R_L sur un ensemble de L données d'entraînement, soit :

$$R_L : f_\theta \rightarrow \frac{1}{L}\sum_{l=1}^{L} Q((x_l,y_l),f_\theta) \qquad (1.12)$$

Grâces à ces caractéristiques de non-linéarité et d'un haute dégrée de connectivité, les MLPs possèdent une grande puissance computationnelle. Mais ces mêmes caractéristiques sont en même temps à l'origine des ces défaillances ; premièrement la non-linéarité et la haute dégrée de connectivité rend l'analyse théorique des MLP difficile. Deuxièmement, l'utilisation des couches cachées rend le processus d'apprentissage difficile à visualiser.

1.7. Apprentissage par la méthode de machine à vecteur de support (SVM)

1.7.1. L'origine de SVM

Les ***Perceptrons*** (introduit par Rosenblatt 1957) qui sont à l'origine des *SVM*, sont des classificateurs linéaires de forme :

$$f_\theta(x) = w \,.\, x + b \quad avec \quad \theta = (w, b) \tag{1.13}$$

et tel que décrit par le théorème suivant (théorème de Cover), pour une série d'hyperplans de dimension d :

$$\{x \mapsto w.x+b, w \in R^d, b \in R\} \tag{1.14}$$

possède une capacité de d + 1. Alors, après avoir choisi une fonction arbitraire Φ, on applique un algorithme de perception sur l'exemple $(\Phi(x_l),y_l)$ au lieu de (x_l,y_l) :

$$f_\theta(x) = \Phi(x_l) \,.\, x + b \tag{1.15}$$

Le problème avec les *Perceptrons* est le fait que les problèmes réels ne sont que rarement linéairement séparables. Les ***Φ-machines*** introduites peu après par Nilsson en 1965 règlent ce problème. L'idée est de d'abord envoyer les vecteurs d'entrées dans un autre espace de caractéristiques (*feature space*) qui contient plus de dimensions que l'espace initial. La figure suivante illustre ce principe :

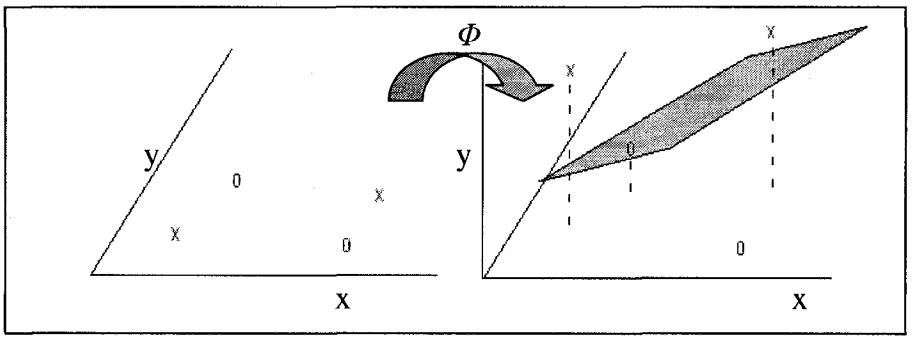


Figure 1.5 : Φ-machines, en ajoutant une dimension une séparation linéaire devient réalisable où le vecteur d'entrée passe de [x] à [x, x^2] (adapté de R. Collobert)

La différence entre les *SVM* avec les *Perceptrons* est que dans le cas des *SVM* la marge entre les deux classes est maximisée (tel que montré dans la figure suivante).

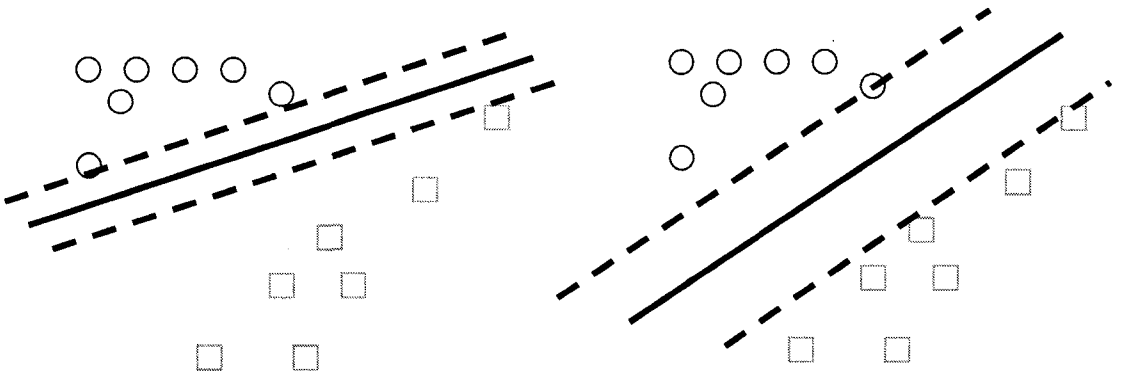


Figure 1.6 : dans le cas (a) la marge qui est la distance entre les deux lignes en tiret a été minimisée (adapté de R. Collobert)

Pour le cas de la régression, il s'agit de trouver le séparateur $f_\theta(x) = w \,.\, x + b$ qui minimise la norme $||w||$ tout en gardant les données d'entraînement (x_l, y_l) à l'intérieur d'une certaine zone que nous définissons par 2ε ($\varepsilon > 0$), nous arrivons alors à l'équations suivante :

$$\forall l \quad \begin{matrix} (\omega \cdot x_l + b) - y_l \leq \varepsilon \\ y_l - (\omega \cdot x_l + b) \leq \varepsilon \end{matrix} \qquad (1.16)$$

En transformant les données d'entraînement (x_l,y_l) en $(\Phi(x_l),y_l)$ et en laissant une marge d'erreur ξ_l et ξ_l^* de chaque coté nous arrivons alors à l'équation suivante :

$$\forall l \quad \begin{matrix} (\omega \cdot \Phi(x_l) + b) - y_l \le \varepsilon + \xi_l \\ y_l - (\omega \cdot \Phi(x_l) + b) \le \varepsilon + \xi_l^*, \end{matrix} \tag{1.17}$$

Le problème revient alors à minimiser aussi ces marges d'erreur, ce qui nous conduit à minimiser la fonction de coût suivant :

$$\frac{\mu}{2}\|w\|^2 + \frac{1}{L}\sum_{l=1}^{L}(\xi_l + \xi_l^*) \tag{1.18}$$

6Où le paramètre μ indique le compromis entre la taille de la marge et la somme des erreurs. Certain auteur remplace μ par C qui est égal à $1/\mu L$.

1.7.2. Calcul de w - b (minimisation d'une fonction Quadratique sous contrainte)

Utilisant la méthode classique de Lagrangien, en introduisant les variables α et α^*, on doit minimiser :

$$(\alpha,\alpha^*) \mapsto \frac{1}{2\mu}\sum_{l=1}^{L}\sum_{l=1}^{L}(\alpha_l^* - \alpha_l)(\alpha_m^* - \alpha_m)\Phi(x_l)\Phi(x_m) - \sum_{l=1}^{L} y_l(\alpha_l^* - \alpha_l) + \varepsilon\sum_{l=1}^{L}(\alpha_l^* - \alpha_l) \tag{1.19}$$

Avec les contraintes suivantes :

$$\begin{matrix} \sum_{l=1}^{L}(\alpha_l - \alpha_l^*) = 0 \\ \forall l \quad 0 \le \alpha_l, \alpha_l^* \le \frac{1}{L} \end{matrix} \tag{1.20}$$

Alors, on peut obtenir la valeur de w et b utilisant les équations suivantes :

$$
\begin{aligned}
& w = \frac{1}{\mu}\sum_{l=1}^{L}(\alpha_l^* - \alpha_l)\Phi(x_l) \\
& (w\cdot\Phi(x_l)+b) - y_l - \varepsilon \qquad quand \quad 0 \le \alpha_l \le \frac{1}{L} \\
& y_l - (w\cdot\Phi(x_l)+b) - \varepsilon \qquad quand \quad 0 \le \alpha_l^* \le \frac{1}{L}
\end{aligned}
\tag{1.21}
$$

1.7.3. Fonction Kernel

Une fonction ***Kernel*** est défini comme une fonction de deux variables qui permet de définir un produit interne d'un espace de caractéristique, autrement dit k(..., ...) est un kernel s'il existe une fonction Φ tel que

$$\forall x_1, x_2 \quad k(x_1, x_2) = \Phi(x_1) \cdot \Phi(x_2) \tag{1.22}$$

Deux exemples des kernels les plus utilisés (tiré de Vapnik 1995) sont celui de kernel ***Gaussien*** :

$$k(x_1, x_2) = \exp(-\|x_1 - x_2\|^2 / (2\sigma^2)), \tag{1.23}$$

et le kernel ***Polynomial*** :

$$k(x_1, x_2) = (1 + x_1 \cdot x_2)^P, \tag{1.24}$$

Les kernels ont été introduit dans le SVM par Boser et al. (1992). L'intérêt des kernels pour les SVM vient du fait que l'on peut remplacer un produit interne $\Phi(x_l) \cdot \Phi(x_m)$ dans un problème SVM par une fonction kernel. Ceci nous permet d'écrire la fonction de décision de SVM de la manière suivante :

$$f_\theta(x) = w \cdot \Phi(x) + b = b + \sum_{l=1}^{L} \alpha_l y_l \Phi(x_l) \cdot \Phi(x) = b + \sum_{l=1}^{L} \alpha_l y_l k(x_l, x) \tag{1.25}$$

Il devient alors possible de lier une entrée à une sortie sans savoir explicitement la fonction de transformation Φ. Ainsi le problème d'apprentissage revient tout simplement à trouver les bonnes valeurs de α_l qui minimise l'équation (1.18).

Grâces à leurs grandes flexibilités, les machines à support de vecteurs ont eu beaucoup de succès dans plusieurs domaines tel que bioinformatiques et reconnaissance de texte. Ils donnent de bonnes performances sur les problèmes de classification de patron. Grâces à l'utilisation des Kernel, les SVM peuvent être construits de façon modulaire ce qui est une caractéristique importante dans

l'ingénierie informatique. Aussi, le fait de travailler avec des espaces de caractéristique de grande dimension permet de résoudre des problèmes complexes.

1.8. Apprentissage par construction d'arbre

Dans les arbres de décision chaque nœud d'un arbre de décision contient un test (un SI...ALORS) et les feuilles prennent des valeurs en conséquence. Voici un exemple d'un arbre simple pour décider de tirer le ballon ou d'avancer vers le but dépendamment de différents paramètres :

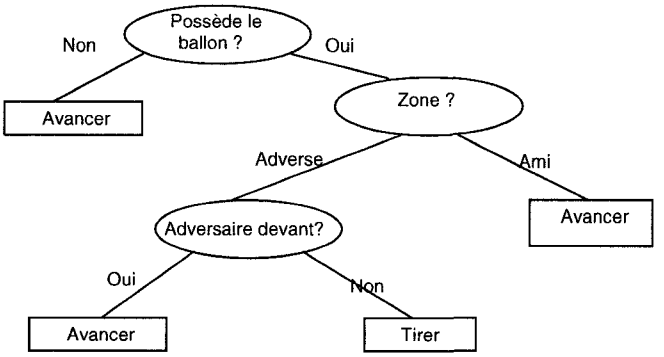
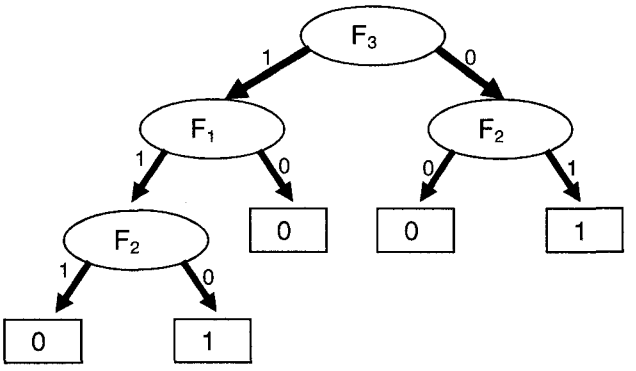


Figure 1.7 : Exemple d'un arbre de décision pour tirer vers le but

Pour construire l'arbre de décision, les algorithmes connus sont ID3 (Quinlan 1986) et C4.5 (Quinlan 1993). Il s'agit de trouver quel attribut tester à chaque nœud. C'est un processus récursif. Pour déterminer quel attribut tester à chaque étape, on utilise un calcul statistique qui détermine dans quelle mesure cet attribut sépare bien les exemples Oui/Non. On crée alors un nœud contenant ce test, et on crée autant de descendants que de valeurs possibles pour ce test. Voici un autre arbre sous forme binaire cette fois-ci :



$$h = (\neg f_3 \wedge f_2) \vee (f_3 \wedge f_1 \wedge \neg f_2)$$

Figure 1.8 : Exemple d'un arbre binaire de décision (adapté de T. Lozano-Pérez)

Où h représente la valeur de la sortie et f_n est une entrée de système. Pour construire un arbre de décision on peut utiliser l'algorithme récursif de *Quinlan* . Cet algorithme est représenté par une fonction qui prend en paramètres une série de données sous forme d'entrée/sorite et crée l'arbre. :

```
ConstuireArbre (Données)
{
        SI (toutes les données ont la même valeur y) ALORS
                ContruireNoeudFinale(y)
        SINON
                Trait := ChoisiMeilleurTrait(Données)
                CréerNoeudInterne(Trait,
                                ConstuireArbre(ChoisirFaux(Donnée, Trait)),
                                ConstuireArbre(ChoisirVrai(Donnée, Trait)))

}
```

Au départ, on vérifie si toutes les valeurs y des éléments sont identiques, ce qui est la condition de base de la fonction récursive. Sinon, on choisi une caractéristique ou *Trait* f_n et on divise les données en deux, une série de données pour lesquelles la valeur du *Trait* f_n est fausse, et une série pour les quelles elle est vraie. On construit alors un nœud interne et on continue la construction de l'arbre pour chaque coté de ce nœud.

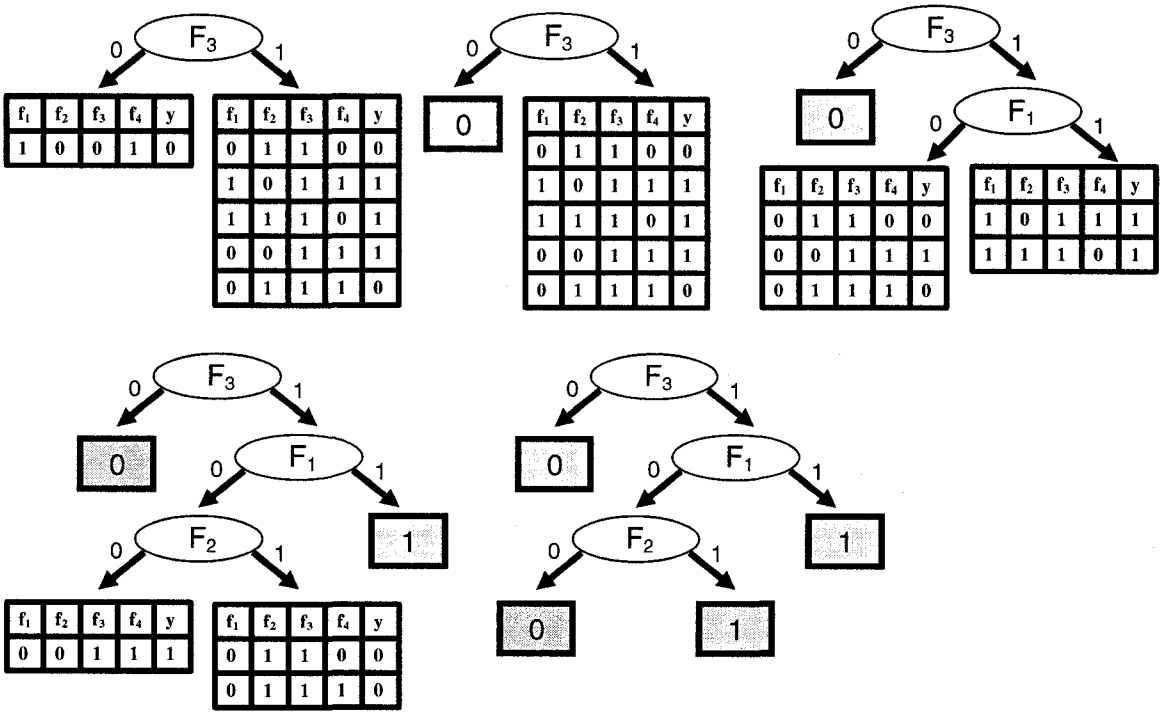


Figure 1.9 : Exemple de construction d'un arbre d'apprentissage (adapté de T. Lozano-Pérez)

La façon de choisir un nœud est importante pour avoir un arbre de taille optimale. Afin d'avoir une taille optimale, on minimise la valeur moyenne de l'entropie de données pour les nœuds entrants. La valeur de l'entropie (H) est calculée grâce à la formule présentée à la figure suivante où p est la fraction positive des exemples dans l'ensemble des données (obtenu lors de la séance d'entrainement) si on utilise l'attribut en question comme classificateur. La figure suivante présente un exemple de ce calcul :

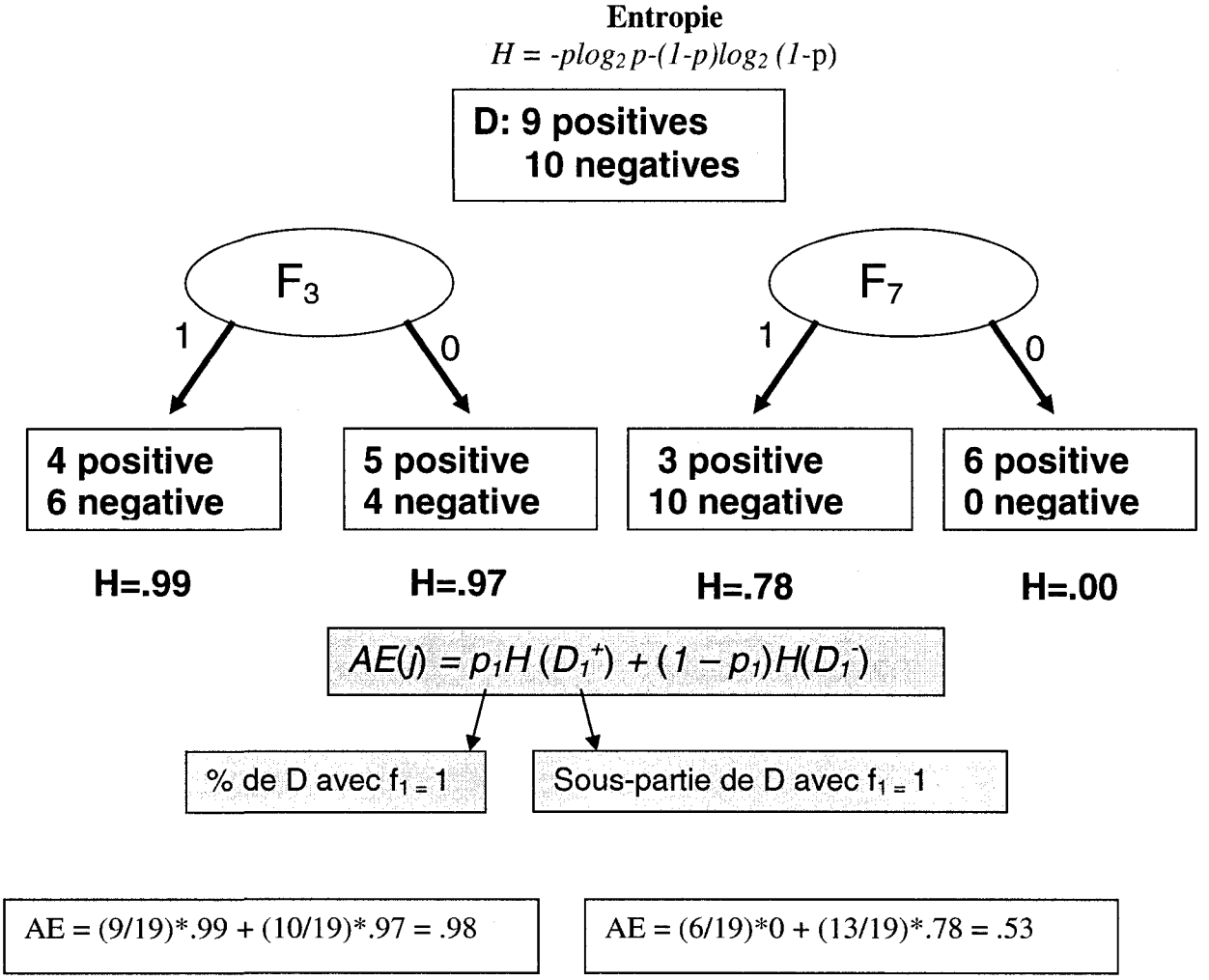


Figure 1.10 : Exemple de calcul de l'entropie moyenne AE (adapté de T. Lozano-Pérez)

L'algorithme de classification calcule la valeur de l'entropie moyenne AE pour chaque attribut et choisit alors celui qui possède la plus petite valeur, c'est à dire celui qui permettra de séparer le plus nettement possible les exemples qui restent. On remarque que dans le cas de l'exemple présenté le choix f_7 sera préférable comme caractéristique pour diviser l'arbre en deux puisque sa valeur calculée d'AE est inférieure à celle calculée pour

Les arbres de décision fournissent des méthodes effectives qui obtiennent de bons résultats dans la pratique. Les arbres de décision possèdent l'avantage d'être compréhensibles par tout utilisateur (si la taille de l'arbre produit est raisonnable) et d'avoir une traduction immédiate en termes de règles de décision. Aussi, comparé aux réseaux neurones, les résultats expérimentaux semblent prouver les faits suivants (F. Denis et R. Gilleron) : le temps de calcul pour les réseaux de neurones est en général supérieur au temps de calcul pour les systèmes basés sur les arbres de décision (le rapport variant entre 1 et beaucoup). Par contre les méthodes de construction d'arbre basées sur de nombreuses heuristiques, ne sont jamais remis en question (pas de retour en arrière) et sont donc non optimales. Aussi, il est possible de modifier les valeurs de nombreux paramètres, de choisir entre de nombreuses variantes et faire le bon choix n'est pas toujours aisé.

1.9. Apprentissage par algorithme de Naïve Bayes

Cet algorithme, étant basé sur la loi d'interférence probabiliste de Bayes, est très efficace pour des systèmes complexes avec des dimensions élevées (paramètres utilisé pour déterminer la sortie du système). Il est à noter que l'approche dite *bayesienne* n'est pas souvent utilisée lorsque le nombre de dimensions est élevé mais le fait de prendre la version *naïve* permet d'augmenter le nombre de dimensions (au détriment cependant de l'optimalité).

Définissions $R_i(x,y)$ comme étant la fraction de fois où la sortie a pris la valeur de y et l'entrée f_i a pris la valeur de x. En se basant sur les valeurs calculées de R_i, on peut alors décider la valeur de la sortie pour une nouvelle paire d'entrée/sortie. En fait, dans ce cas on compare la probabilité que la sortie ait une valeur plutôt qu'une autre et on choisit la valeur pour laquelle on a eu la plus grande probabilité. Voici un exemple binaire de la version la plus simple de cet algorithme.

f_1	f_2	f_3	f_4	y
0	1	1	0	1
0	0	1	1	1
1	0	1	0	1
0	0	1	1	1
0	0	0	0	1
1	0	0	1	0
1	1	0	1	0
1	0	0	0	0
1	1	0	1	0
1	0	1	1	0

$R_1(1,1) = 1/5$ $R_1(0,1) = 4/5$
$R_1(1,0) = 5/5$ $R_1(0,0) = 0/5$
$R_2(1,1) = 1/5$ $R_2(0,1) = 4/5$
$R_2(1,0) = 2/5$ $R_2(0,0) = 3/5$
$R_3(1,1) = 4/5$ $R_3(0,1) = 1/5$
$R_3(1,0) = 1/5$ $R_3(0,0) = 4/5$
$R_4(1,1) = 2/5$ $R_4(0,1) = 3/5$
$R_4(1,0) = 4/5$ $R_4(0,0) = 1/5$

Figure 1.11 : Algorithme Naîve Bayes (adapté de T. Lozano-Pérez)

Pour une nouvelle série de x = <0, 0, 1, 1>, afin de déterminer la valeur de la sortie *y* (0 ou 1), on calcule S(y) :

$$S(1) = R_1(0,1) * R_2(0,1) * R_3(1,1) * R_4(1,1) = .2$$
$$S(0) = R_1(0,0) * R_2(0,0) * R_3(1,0) * R_4(1,0) = 0$$

Donc, on décide de prendre la valeur de 1 pour la sortie dans ce cas puis que c'est plus probable que la vrai valeur de *y* soit 1 que 0 car S(1) a une valeur supérieure à S(0).

1.10. Apprentissage par renforcement

Cette approche est fondée sur le concept psychologique « une récompense immédiate d'une action après son occurrence augmente sa probabilité d'occurrence, et une punition d'une action diminue sa probabilité d'occurrence » (Thondike 1911). C'est une méthode numérique, inductive et continue.

On peut résumer cette approche avec l'algorithme suivant :

Algorithme général

- Initialiser l'état interne S
- Répéter :
  a. Observer la situation x du monde,
  b. Choisir l'action *a* à exécuter à partir de la fonction d'évaluation
  $a = FctEval(x,T)$, T représentant une part de hasard,
  c. Exécuter l'action *a* dans le monde,
  d. Soit r le signal de renforcement associé à l'action *a*, mettre à jour l'état S :
  $S_{nouveau} = FctMiseAJour(S_{ancien},x,a,r)$

Figure 1.12 : L'algorithme général de méthode de renforcement (tiré de E. Zenou)

Donc, par exemple pour un robot joueur de soccer, il va d'abord observer la situation x autour de lui soit la position des autres robots et le ballon, il va ensuite choisir une action a (soit faire un tir, faire un passe, avancer vers le but ou avancer vers le ballon) en se basant sur la situation x et à partir de la fonction d'évaluation, une fois l'action réalisée il va recevoir un signal de renforcement r qui peux par exemple être basé sur l'avancement vers le but adverse. Ensuite, la fonction de la mise à jour nous fournira le nouvel état S. Lorsque le signal de renforcement r provient de l'agent lui-même, il est dit intrinsèque. Le but de l'agent est d'apprendre pour chaque état les actions maximisant la somme de ses récompenses espérées.

La difficulté dans ce type d'apprentissage est que la récompense peut ne venir qu'à la fin. Par exemple, le robot peut recevoir une récompense une fois qu'il a effectué son tir selon le fait que ceci a causé un avancement du ballon vers le but adverse ou non (un rebondissement ou un tir dans le mauvais sens). Auquel cas, le robot doit essayer de comprendre quelles actions ont mené à ce résultat. C'est le problème de l'assignation du crédit (à quelles actions associer la récompense).

Les questions fondamentales concernant cette approche sont souvent les suivantes :

- À quelle vitesse doit se faire l'apprentissage? Un apprentissage doit se faire en fonction de changement de l'environnement de l'agent et un apprentissage trop lent ne vaut pas la peine des calculs supplémentaires introduits dans le système.

- Comment faut-il approximer les fonctions du contrôle? Utiliser un « look-up table », une approximation continue ou une approximation discrète

- Quel algorithme faut-il utiliser? Deux types d'algorithmes sont très connus, soient:
    - *AHC* ou « *Adaptive Heuristic Critic* » : avec cette méthode au lieu d'essayer de maximiser une récompense de façon instantanée, on essaie de maximiser la valeur heuristique *v*, qui est calculée par un *critique*. Cette valeur est ensuite utilisée de façon locale pour aider la composante d'apprentissage par renforcement (RL) à choisir une action appropriée en considérant l'état et la politique choisie est celle qui est utilisée à ce moment dans la composante (RL). La figure suivante illustre cette approche où s est l'état de l'agent, r est la récompense instantanée et a est l'action choisie :

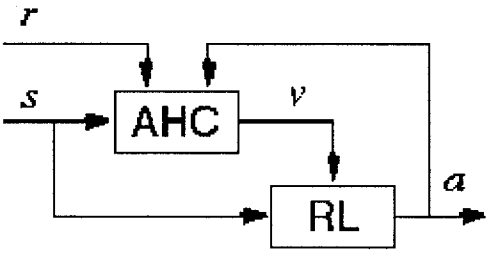


Figure 1.13 : Approche AHC[1]

    - « *Q-learning* » : où une seule fonction *Q* (ou une fonction d'évaluation) évalue les actions et les états. Elle représente la prévision sur la somme des renforcements que l'agent espère recevoir en exécutant une action *a* à partir d'une situation *x*. En d'autres termes, on peut dire que la fonction *Q* cherche à déterminer pour une situation donnée, l'action qui donne la plus grande valeur pour la fonction d'évaluation. Des différentes études ont démontré que cette méthode est supérieure à la méthode *AHC* pour des applications réactives et elle domine en ce moment les approches basées sur la récompense des systèmes robotiques.

[1] http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume4/

Algorithme Q-learning

1. Initialiser l'état interne S
2. À chaque instant t :
   a. Observer la situation x du monde
   b. Choisir l'action a à exécuter à partir de la situation xt utilisant la fonction Q :
      $a = FctEval(x,Q(x,a),T)$
   c. Exécuter l'action a
   d. Observer la nouvelle situation x'. Soit r le signal de renforcement associé à l'action a, mettre à jour l'état S et la fonction d'évaluation Q :
      $S_{nouveau} = FctMiseAJour(S_{ancien},x,a,r)$
      $Q_{nouveau} = FctMiseAJour(Q_{ancien},x,a,r)$

Figure 1.14 : Algorithme Q-Learning (adapté de E. Zenou)

1.11. Apprentissage évolutionnaire

Différents types d'algorithmes sont connus actuellement sous l'appellation d'algorithmes d'évolution: les algorithmes génétiques, les stratégies évolutives (*evolution strategies*) et la programmation évolutive (*evolutionary programming*). Nous examinons dans cette section les algorithmes génétiques.

1.11.1. Apprentissage par algorithme génétique

Les algorithmes génétiques sont des outils d'optimisation qui ont été aussi utilisés pour déterminer le comportement d'un système multi robot. Ayant été inspiré de la biologie, leur fonctionnement est fondé sur des mécanismes d'évolution. Normalement, on applique cet algorithme à une population qui évolue le long de plusieurs générations. Chaque agent (individu) est caractérisé par un génotype, qui définit une partie ou toute sa structure. Une première population de génotypes est générée de façon aléatoire. Lors du déroulement de l'algorithme on évalue les

performances de chaque agent selon des critères prédéfinis qui détermine leur bonne forme ou leur « fitness ». La fitness de chacun individu est évaluée en fonction de sa performance à chaque itération. En fonction de celle-ci, certains individus produisent de nouveaux individus utilisant trois opérations ; en combinant leur génome avec celui d'autres agents de la population, ce qui constitue une opération de *recombinaison* ou « cross-over », en modifiant aléatoirement le génome résultant, ce qui constitue une opération de *mutation* ou en faisant reproduire seulement les génomes supérieurs et éliminant les individus qui résultent une performance inférieure, ce qui est intitulé une opération de *reproduction* La recombinaison, la mutation et la reproduction constituent les opérateurs génétiques de l'algorithme génétique. Voici un schéma qui illustre de façon détaillée un algorithme génétique:

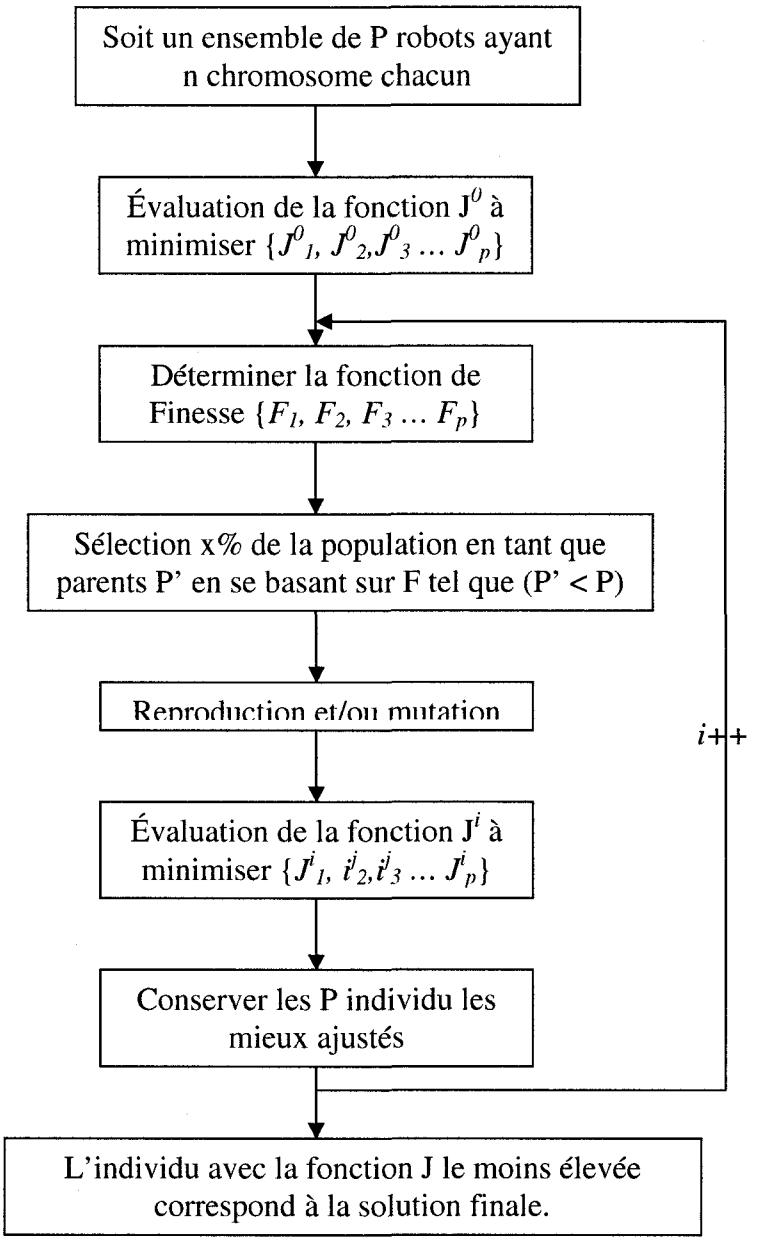


Figure 1.15 : Apprentissage par algorithme génétique

Prenons comme exemple un robot joueur de soccer qui veut apprendre à éviter les obstacles utilisant un algorithme génétique. On peut définir des différents états du système soit : blocage, pas d'obstacle, obstacle à gauche, obstacle à droite, obstacle en face, et obstacle en arrière. Et on aura les 4 différentes comportements

élémentaires suivants : avancer, reculer, aller vers la gauche ou aller vers la droite. Le chromosome d'un robot peut alors être la concaténation (une chaîne) de N mots dans un sous-espace de l'espace binaire $\{0,1\}^M$. où N est le nombre d'états et M le nombre de sorties (comportements), on aura alors un chromosome qui ressemble à ceci :

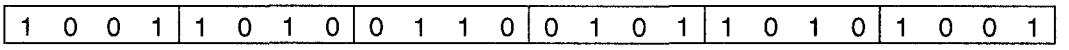


Figure 1.16 : Chromosome d'un robot pour éviter des obstacles

La population de robots va alors évoluer en suivant l'évolution des chromosomes sous l'action des opérateurs génétiques, et chaque robot peut s'auto-évaluer en fonction de la distance parcourue sans être bloqué et il va aussi être conscient de la performance des autres robots. On peut alors décider de faire une mutation d'un robot si son indice de performance est assez bas et il n'a pas muté récemment, ou on peut faire un croisement des indices entre deux robots en se basant sur leur performance.

Pour des problèmes où il y a un grand nombre d'entrées et il y a de nombreux minima locaux les algorithmes génétiques sont bien adaptés. Par contre, un bon codage des paramètres dans le génotype et une fonction d'évaluation efficace sont nécessaires. La fonction d'évaluation peut servir à optimiser un aspect de la mission ou combiner tous les aspects ensemble. Par exemple, Arkin suggère trois classes de fonction de fitness :

- Sécuritaire : optimisé pour éviter le plus possible la collision avec des obstacles en se dirigeant quand même vers le but,
- Rapide: optimisé pour avancer vers le but en moins de temps possible,
- Direct : optimisé pour prendre le chemin le plus court vers le but.

Pour faire évoluer un système de contrôle de façon continu ou « *On-Line* », il faut considérer le contrôleur comme une population qui contient des processus de

comportement concurrent. La fonction de performance et la contribution de chaque processus sont est évaluées sur une période de temps défini, et les résultats donnés guident la reproduction de ce comportement. Des études plus poussées (en simulation) ont aussi été abordées afin de faire évoluer non seulement le contrôleur d'un système robotique mais aussi sa morphologie (Sims 1994).

1.12. Apprentissage par des contrôleurs flous ou « Fuzzy »

Un contrôleur flou est basé sur une logique floue. La logique floue, contrairement au prédicat logique conventionnel qui prend soit une valeur vrai ou faux, permet de déterminer à quel point une variable appartient à un ensemble flou (défini par une fonction d'appartenance). Ces variables ne vont alors pas changer de façon brusque. Les fonctions d'appartenance mesurent de façon numérique le degré d'appartenance d'une variable à son ensemble flou associé. L'architecture d'un système de contrôle flou consiste en éléments montrés dans la figure suivante :

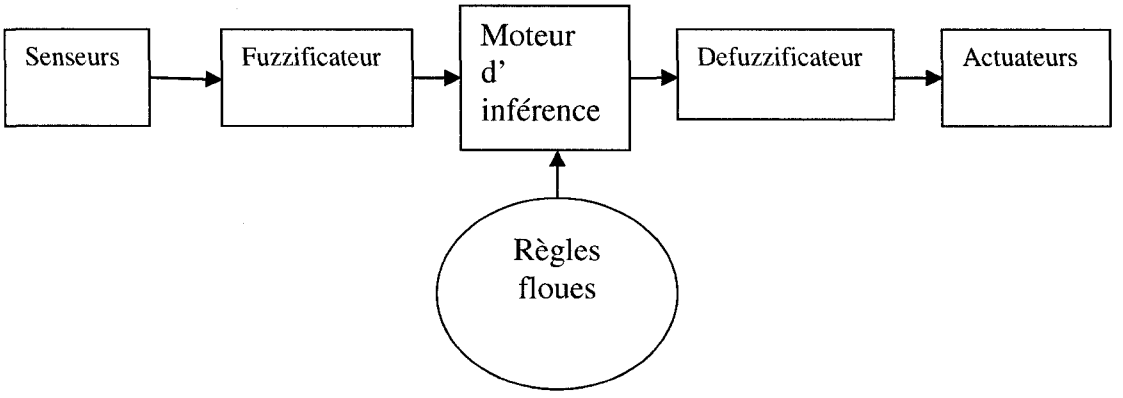


Figure 1.17 : Architecture du système de contrôle flou (traduit de R.C. Arkin)

- Le « *Fuzzificateur* » qui lit les données de l'entrée de système et les associe à des valeurs floues.
- Les « *Régles Floues* » contiennent des lois de base de type « SI (une condition) ALORS (une action) ».
- « *Engin d'inférence Fuzzy* » qui associe les ensembles flous d'entrée et de sortie ensemble en se servant des fonctions d'appartenance et les lois de base.
- Et finalement le « *Defuzzificateur* » qui associe un ensemble de sorites floues aux sorties de l'actuateur.

La figure suivante montre un exemple de contrôleur flou pour une variable de changement de direction d'un robot. Les fonctions d'appartenance d'entrée sont présentées verticalement et celles de sortie sont présentées horizontalement et les lois de base qui les associent ensemble sont au milieu.

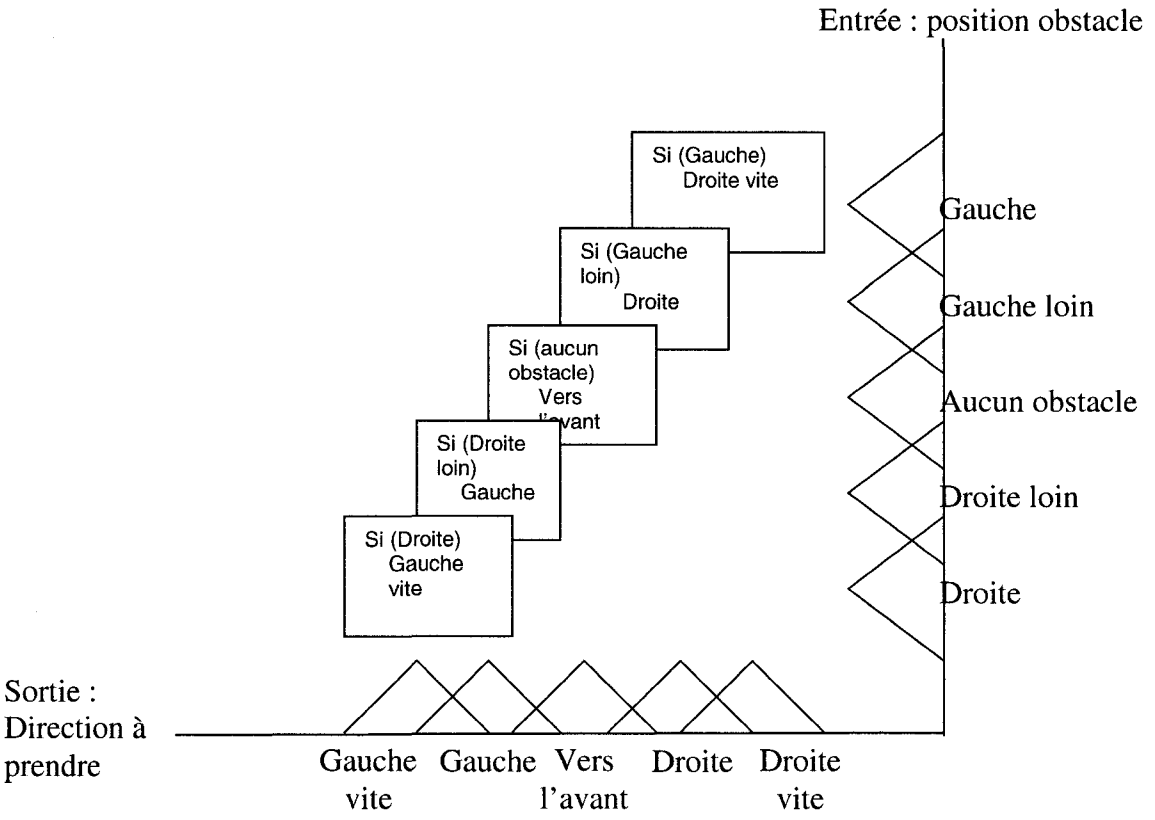


Figure 1.18 : Exemple d'un système de logique floue

L'apprentissage dans ce cas consiste à générer les lois floues de base en appliquant les étapes suivantes :

- Le concepteur fourni un modèle qui représente les fonctions d'entrée et les lois stratégiques de base qui définissent les réactions à un stimulus de façon qualitative.

- Le système crée alors un squelette pour les lois de bases en s'assurant que ceci couvre complètement toute l'espace de stimuli-réponse et initialise les fonctions d'appartenance de sortie.
- En se servant d'une série de règles spécialisées, les fonctions d'appartenance sont modifiées afin de diminuer l'erreur entre la valeur de sortie déterminée par le contrôleur flou et la valeur désirée de la sortie (obtenu lors d'une séance d'entraînement) pour arriver à des relations plus appropriées (normalement, le critère utilisé est de diminuer l'erreur moyenne au carrée).

1.13. Apprentissage par expérience

Un apprentissage par expérience essaie de créer un modèle qui prédit la réponse d'un système utilisant des résultats obtenus lors des expériences antérieures et en utilisant certains paramètres de calcul de réussite pour les distinguer.

1.13.1. Basé cas

Dans ce cas, les expériences sont sauvées en tant que cas structurés.
L'algorithme de base de cette approche est le suivant (Arkin)

- Classifier le problème courant,
- Utiliser la description du problème afin de trouver un cas similaire dans la mémoire,
- Adapter la solution de l'ancien cas à la nouvelle situation,
- Appliquer la nouvelle solution et évaluer le résultat,
- Apprendre par sauvegarde du nouveau cas et ses résultats.

1.13.2. Basé mémoire

C'est un cas extrême de l'apprentissage basé cas, dans lequel des détails numériques explicites de chaque expérience sont sauvés en mémoire. Cette méthode est très efficace pour apprendre des approximations de loi de contrôle pour des tâches telles que le balancement de poteau, jouer au billard ou la jonglerie (Atekson, Moore et Schaal 1997).

L'apprentissage par la méthode de la moyenne locale et le voisin le plus proche (Near Neighbor and Local Averaging) est un exemple de l'apprentissage basé mémoire. Cette méthode est sans doute la plus simple pour faire l'apprentissage. Elle base la prédiction d'une valeur de sortie sur les

anciennes valeurs qui ont des entrées les plus semblables. Par exemple, un système qui doit décider la sortie y dépendamment de la perception de système noté par x.

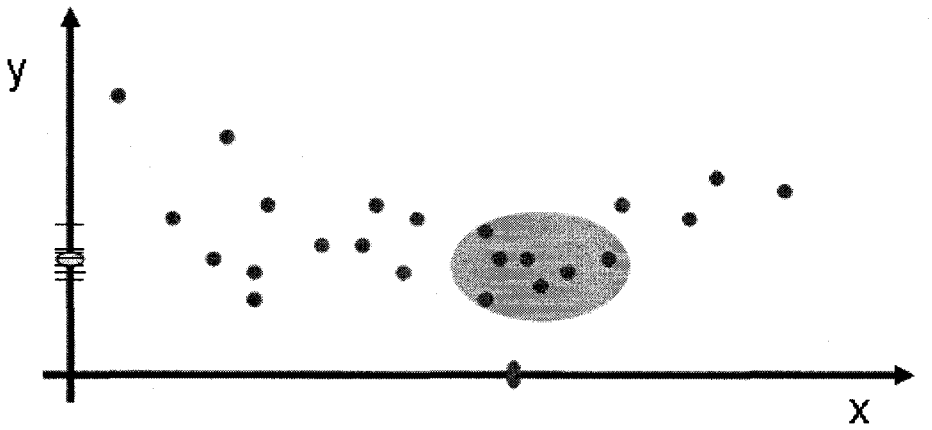


Figure 1.19 : Apprentissage par moyenne locale/voisin le plus proche (adapté de T. Lozano-Pérez)

En trouvant une série de données enregistrées en mémoire appartenant à une région limitée par les contraintes prédéfinies (dans ce cas l'ellipse), on peut affecter la valeur moyenne de ces données pour prédire la valeur de y. Un problème avec cet apprentissage est le fait que cet apprentissage utilise tous les attributs d'un cas à chaque fois pour calculer la similarité avec un nouveau cas à classer.

1.14. Apprentissage multi stratégies

On peut combiner deux ou plusieurs des approches présentées précédemment afin de construire un mécanisme plus robuste. Par exemple, les réseaux de neurones et les algorithmes génétiques peuvent être utilisés ensemble; Samuel Delepoulle et al. (2001) combine le réseau neurone avec apprentissage génétique où les coefficients du réseau de neurone sont mis à jours via un apprentissage génétique.

En d'autres termes, l'apprentissage n'est pas déterminé par les gènes mais les variables qui sont codées génétiquement. Ces variables interviennent dans la modification des réponses du réseau au cours du temps. En fait, l'apprentissage n'est pas entièrement prédéterminé génétiquement mais reste au contraire sous l'influence de l'environnement. L'« Apprentissage » ici consiste en modifications de l'activation du réseau en fonction des stimulations qu'il a reçues de l'environnement. Ceci consiste en une modification des poids du réseau. Ces modifications ne sont pas purement déterministes : une connexion du réseau est choisie aléatoirement et son poids est modifié en fonction des neurones auxquels elle est connectée.

Dans ce cas, la valeur de récompense n'est pas une récompense pour le réseau lui-même ; cette valeur est utilisée pour déterminer la performance de l'agent et donc sa probabilité de survie à la génération suivante, mais pas pour que le réseau tente de corriger ses comportements inadéquats au cours de sa vie, comme dans un apprentissage par renforcement.

1.15. Comité de machines (Committee Machines)

Un principe qui est souvent utilisé en ingénierie en général pour simplifier les problèmes est le principe de la division pour conquérir. Ce principe consiste à résoudre une tâche complexe computationnelle en la divisant à plusieurs tâches computationnelles simples et en combinant leurs résultats à la fin. En apprentissage supervisé, la simplicité computationnelle est achevée en distribuant les tâches d'apprentissage vers un certain nombre d'éléments intitulés des ***experts***. La combinaison de ces experts constitue un comité de machines (***Committee Machine)*** qui va combiner les connaissances acquises par les experts et arriver à une solution commune. On distingue deux catégories de comité de machines:

1. *Structure statique* : dans cette catégorie les réponses des experts sont combinées ensemble par un mécanisme de calcul qui n'inclut pas directement

les signaux d'entrée d'où la désignation statique. Cette catégorie inclut les méthodes suivantes :

- *Moyenne d'ensemble (Ensemble averaging)* : où les sorties des différents experts sont linéairement combiné pour produire la sortie finale.
- *Stimulateur (Boosting)* : où un algorithme d'apprentissage faible est convertie en un qui possède plus de précision.

2. *Structure dynamique* : dans cette catégorie les signaux d'entrée sont directement inclus dans le calcule des réponses des experts et le comité de machines utilise un intégrateur pour combiner les résultats ensemble afin d'arriver au résultat final. On distingue deux types de structure dynamique :
   - *Mélange des experts* : où la division de l'espace et la recombinaison des réponses des experts se fait par un modèle, nommé pondérateur, dépendant des entrées. Les réponses des experts sont combinées de façon non-linéaire utilisant un seul pondérateur dans ce cas. Alors, le principe de la division pour conquérir n'est utilisé qu'une fois.
   - *Mélange des experts hiérarchique* : où les réponses des experts sont combinées de façon non-linéaire utilisant plusieurs pondérateurs arrangés de façon hiérarchique. Alors, le principe de la division pour conquérir est utilisé plusieurs fois.

1.16. Conclusion : choix de méthodes d'apprentissage utilisées

Dans cette section une analyse des méthodes d'apprentissage les plus utilisées et populaires parmi celles présentées précédemment est effectuée afin de choisir celles qui peuvent être considérées comme étant les plus intéressantes à étudier dans le cadre de ce projet.

L'**apprentissage par la méthode de la moyenne locale et le voisin le plus proche** a l'avantage d'être très simple, et elle est très rapide et efficace pour les systèmes avec peu d'entrées, mais pour des systèmes complexes elle devient moins commode, puis qu'elle requière une représentation des données en plusieurs dimensions (dans ce cas tous les points deviennent éloignés l'un par rapport à l'autre). Cet apprentissage utilise tous les attributs d'un cas à chaque fois pour calculer la similarité avec un nouveau cas à classer.

L'**apprentissage par algorithme génétique** a aussi un taux d'apprentissage lent et ne sera intéressant qu'en tant qu'une méthode combinatoire avec une autre méthode d'apprentissage plus rapide.

La méthode d'**apprentissage par machine à vecteur de support ou SVM** est une méthode récente et peut aussi être applicable dans notre contexte. Elle sera donc étudiée dans le cadre de ce travail de maîtrise.

Entre les méthode populaires de **réseau de neurones**, et **l'apprentissage par construction d'arbre**, la méthode MLP (une variante de réseau de neurone) a été choisi, vue que comparé à l'apprentissage par la construction d'arbre la procédure générée par un réseau de neurones est généralement légèrement meilleure et les réseaux de neurones semblent mieux se comporter en présence de bruit. Il semble

enfin que les réseaux de neurones se comportent mieux que les arbres de décision lorsque tous les attributs sont signifiants (Denis F., Fillero R 2008).

La méthode d'**apprentissage basé sur mémoire** étant simple et populaire sera aussi étudié dans le cadre de ce travail afin d'apprendre le meilleur patron à utiliser pendant une phase de jeu donnée.

Chapitre 2
Problématique et plateforme utilisé

2.1. Domaine d'utilisation

Le domaine d'utilisation du système conçu dans le cadre de ce travail de recherche est celui d'une équipe de véhicules autonomes synchronisés avec les caractéristiques suivantes:

- Il existe une équipe constituée de véhicule autonome qui collabore pour atteindre un but commun,
- Les agents peuvent communiquer leurs états et leurs décisions entre eux périodiquement, sois sans aucune restriction de communication (pendant que l'équipe est *off-line* par exemple lors des pauses) ou avec une certain compromis entre la performance et la communication (pendant que l'équipe est on-line par exemple quand les robots jouent).

Mentionnons quelques exemples qui font partie de ce domaine : systèmes multi-robots de maintenance des hôpitaux et des usines (Decker 96), systèmes multi-robots utilisés dans les missions de « recherche et sauvetage », missions utilisés dans les champs de bataille (Tambe 97), ou encore missions pour des systèmes composés de plusieurs vaisseaux spatiaux (Stone 97).

2.2. Plateforme utilisée

Le soccer robotisé constitue la plateforme d'essai utilisée dans le présent projet. La problématique de développer une équipe de robots joueurs de soccer autonomes combine le développement de systèmes complexes dans plusieurs domaines tel que l'informatique temps réel, l'intelligence artificielle, les systèmes de perception, contrôle de bas et de haut niveau, les mécanismes de coopération, etc. Ainsi les contributions qui peuvent provenir du soccer robotisé pour d'autres systèmes multi-robots coopératifs sont nombreuses.

Aussi, le soccer ayant été le sport le plus populaire au monde depuis plusieurs décennie, les compétitions mondiales de soccer robotisé telle que la *RoboCup* offrent non seulement un environnement compétitif mais aussi de très bonnes occasions de rassembler la communauté scientifique internationale qui travaille dans des domaines qui y sont reliés. La communauté de la *RoboCup* regroupe environ quatre milles chercheurs provenant de plus d'une trentaine de pays.

La *RoboCup* est une compétition mondiale qui permettant à des équipes de robots de différentes universités ou entreprises de s'affronter. La *RoboCup* a comme l'objectif ultime de: « By 2050, develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer ».

Des ligues différentes se distinguant par le format du terrain et le format des robots sont mises en place dans cette compétition. La « Middle-Size Robot League » est la ligue dans la quelle nos robots participent. Cette ligue permet deux équipes de 4 à 6 robots de s'opposer sur un terrain 8x12m. La figure suivante présente une image d'un match typique de cette ligue.

Figure 2.1: Image d'un match de la RoboCup « Middle Size Robot League ».

2.2.1. Présentation de la plateforme d'essai

Le système développé dans le cadre de ce projet compte 6 robots en ce moment dont 4 sont d'une ancienne génération (la génération basée sur un architecture à vitesses différentielles) et deux d'une nouvelle génération (la génération de robots omnidirectionnels) qui ont été fabriqués au courant des dernières deux années.

Le système de jeu de soccer robotisé est composé de plusieurs éléments :

- une plateforme mécatronique,
- Les modules électroniques,
- Un logiciel de contrôle temps-réel,
- Un simulateur.

2.2.2. Structure modulaire

La figure suivante présente la structure modulaire de l'architecture de contrôle local tel que conçu en grande partie par Julien Beaudry et Sylvain Marleau.

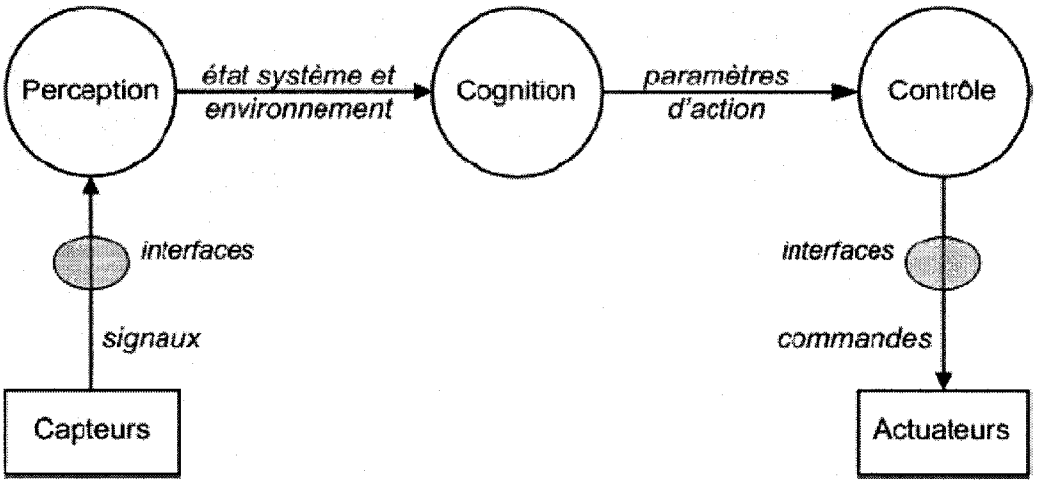


Figure 2.2 : Structure modulaire de l'architecture de contrôle. (Adapté de J. Beaudry)

2.2.3. Module de perception

Le système de perception d'un robot est composé de capteurs. Utilisant des circuits d'interfaces et pilotes, les capteurs transmettent leurs signaux au module de perception qui à son tour après avoir acquis les informations des capteurs, modélisera le système robotisé et son environnement. En fait ce module fait un traitement de l'image provenant des capteurs et met à jour l'état du système et son environnement de façon continue.

2.2.4. Module de cognition

Le module de cognition est l'équivalent d'un cerveau chez les humains. Il utilise l'information qui lui est fourni par le module de perception. Ce module étant assez complexe, son rôle varie en fonction de l'application du système robotisé. C'est dans ce module que toute la partie d'intelligence artificielle ou apprentissage machine se trouve. En effet, ce module s'occupe de la mise à jour des paramètres d'action en ce servant des informations reçues, du système de perception.

2.2.5. Module de contrôle

Le module de contrôle à son tour permet au système robotisé d'agir sur son environnement grâce aux actuateurs du système. Ce module utilise alors les paramètres d'action qui lui sont commandés par le module de cognition et après avoir traité ces informations et avoir effectué un calcul de signaux de commande, il transmet ces signaux de commande aux actuateurs.

2.2.6. Le module de communication

Il ne faut pas oublier que les relations entre les différents modules ne sont pas forcément aussi simplement définies, alors le module de communication est un autre élément important de ce système.

2.2.7. Plateforme électromécanique

Tel que mentionné dans les sections précédentes, la configuration des robots de la première génération est une plateforme à vitesses différentielles symétrique. Il y existe deux moteurs de direction et de propulsion qui sont reliées à deux roues motrices qui se trouvent au centre du robot. Afin d'assurer l'équilibre du robot deux roues libres sont aussi ajoutées à ces

robots. Cette génération de robots peut atteindre les vitesses de 3m/s et 18rad/s. Ils ont été conçus en se basant sur le robot *SpinoS* développé à Polytechnique par Julien Beaudry. Cette plateforme possède aussi des rouleaux de mousse passifs à l'avant et à l'arrière du robot pour contrôler le ballon et un système pneumatique permettant de botter le ballon (voir figure 2.3).

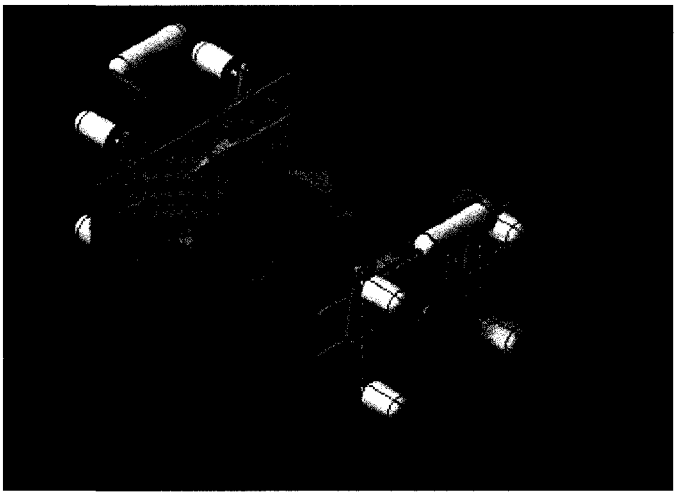


Figure 2.3 : Développement de la plateforme électromécanique des robots. (Tiré de J. Beaudry)

La nouvelle génération est basée sur une architecture omnidirectionnelle. Cette plateforme peux atteindre une vitesse de 5m/s pour le premier robot et jusqu'au 8m/s pour le deuxième robot en développement. Contrairement à la première génération, on a trois degrés de liberté avec cette plateforme en tout temps. La figure 2.4 montre la disposition des moteurs pour le premier robot omnidirectionnel conçu.

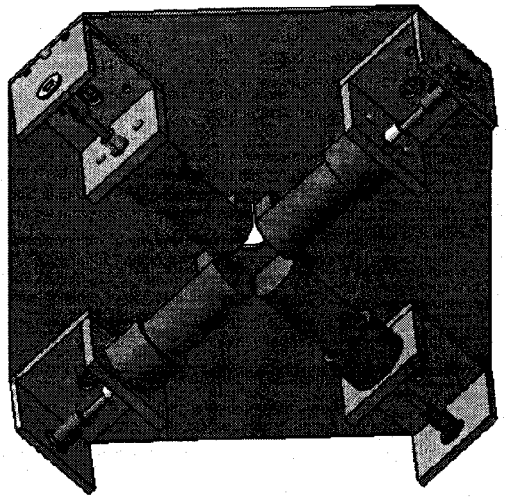


Figure 2.4 : Disposition des moteurs du premier robot omnidirectionnel (Tiré de R.-Commisso M.-A., Béliveau M)

Un deuxième robot omnidirectionnel a été conçu avec une nouvelle configuration cinématique illustré à la figure 2.5.

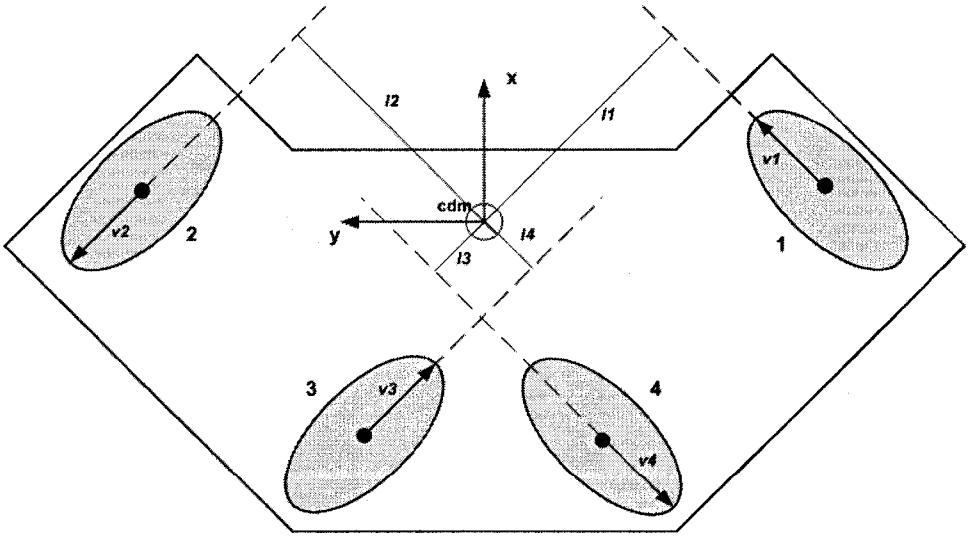


Figure 2.5 : Configuration cinématique du deuxième robot omnidirectionnel

2.2.8. Composantes électroniques

Un ordinateur embarqué est utilisé sur chaque robot. Cet ordinateur de format Half-Size SBC possède les caractéristiques suivantes :

- un processeur de génération Intel Pentium III,
- 256Mo de mémoire SDRAM,
- un contrôleur Ethernet intégré.

Chaque robot possède aussi un lien de communication utilisant la norme IEEE 802.11b sans fil branché à la prise Ethernet de l'ordinateur. Ceci permet de communiquer avec le serveur d'équipe ainsi que les autres robots.

L'interfaçage avec les cartes additionnelles se fait via des bus PC/104 et PC/104+ ainsi que d'autres interfaces courantes tel que RS-232 et USB.

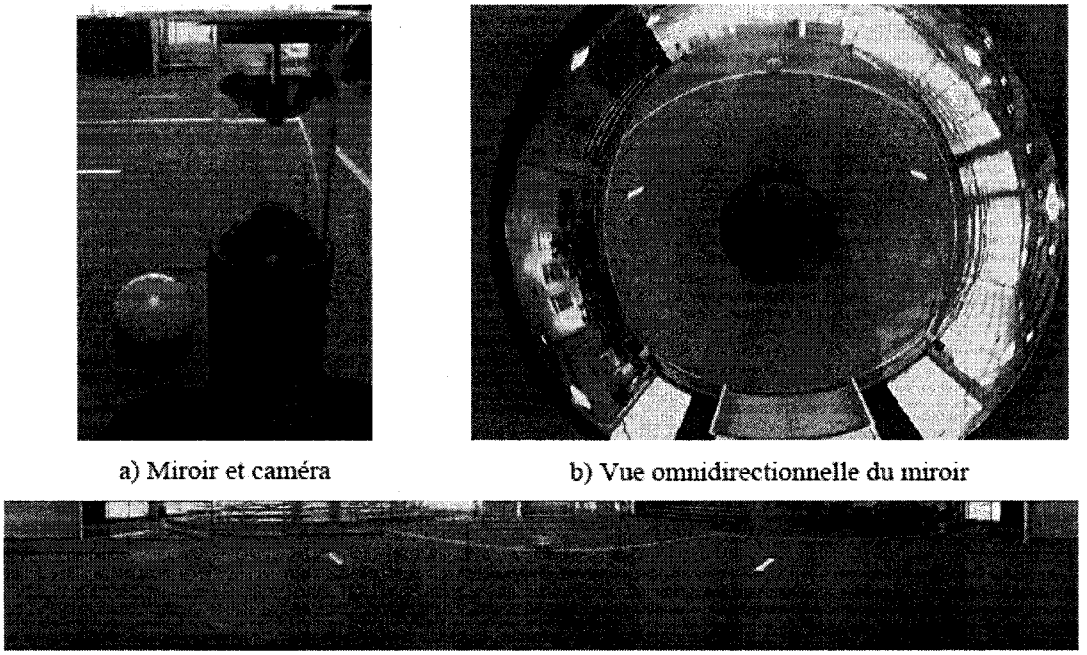
Une carte de contrôle ESC629 de la compagnie RTD est utilisée en tant que le contrôleur des moteurs et le déclencheur de botteur pneumatique. Cette carte utilise un PID numérique avec deux boucles de rétroaction.

Un circuit de commutation des valves du système pneumatique ainsi qu'un un circuit d'alimentation et de recharge automatique des batteries sont aussi parmi les autres composantes électroniques utilisées dans les robots.

2.2.9. Système de perception

Un système de perception est utilisé pour retrouver la position des robots et du ballon sur le terrain. Le système de vision omnidirectionnel utilisé sur chaque

robot est constitué d'un miroir convexe ainsi qu'une caméra USB. L'image suivante présente une image omnidirectionnelle vue de la caméra et l'aspect physique de ce système de vision.



a) Miroir et caméra b) Vue omnidirectionnelle du miroir

Figure 2.6 : Système de perception (Tiré de S. Marleau)

2.2.10. Logiciel de contrôle

Afin de bien contrôler les robots et leur permettre d'agir rapidement et de façon appropriée en temps réel un logiciel de contrôle a été mis en place pour les robots footballeur utilisant le langage de programmation de C++. Utilisant un système d'exploitation de *Debian Linux* (noyau 2.4.18), pour donner une performance satisfaisante l'ordonnanceur du système a été configuré pour fonctionner à 1kHz. Ce logiciel multi-tâches divise les différents modules du robot. On y retrouve quatre classes : *Motion Control, Brain, Perception*, ainsi que *Controller.*

En fait, on y retrouve entre autres deux boucles temporelles principales, une pour les modules de cognition et de contrôle qui fonctionne avec une période de 20ms donc 50Hz et une autre pour la vision au sein du module *Perception* qui fonctionne à 30 images par seconde donc 30Hz. La structure de ce logiciel est illustrée à la figure 2.7.

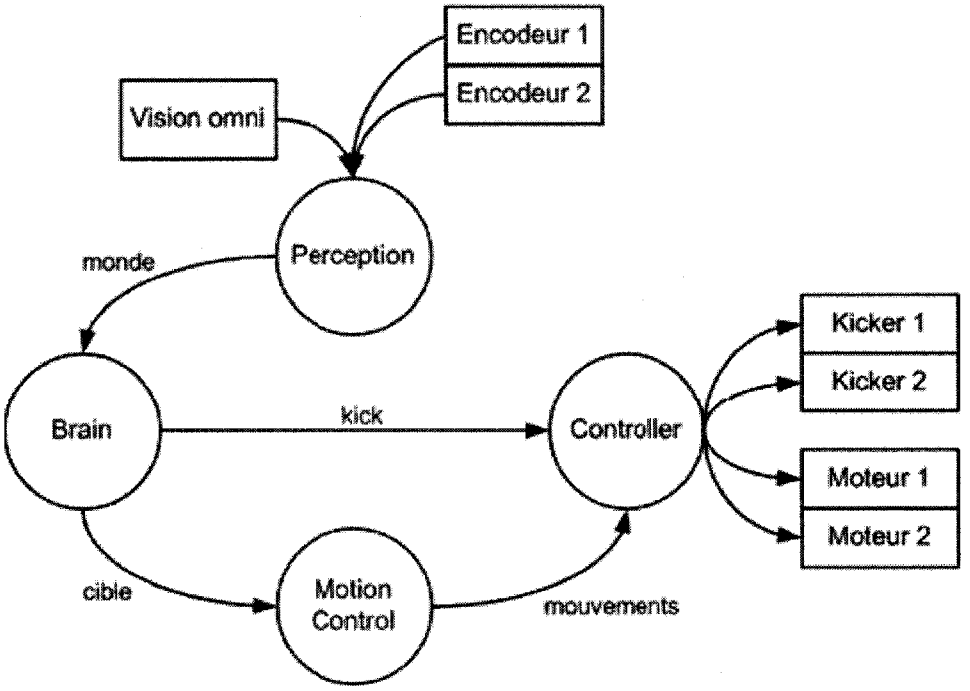


Figure 2.7 : Logiciel de contrôle des robots footballeurs (Tiré de J. Beaudry)

2.2.11. Mécanisme décisionnel

Afin de comprendre l'architecture utilisée par le mécanisme de décision des robots, il est important de définir les différents types d'architectures recherchés. Le tableau suivant, tiré de l'ouvrage de R.C. Arkin présente le spectre des structures de contrôle utilisées pour robots mobiles.

On y retrouve deux extrêmes. Le premier extrême est l'architecture exclusivement réactive permettant une réponse temps réel, déterministe puisque l'on y utilise un traitement simple sans tenir compte de l'environnement qui entoure le robot. Mais, tel que l'on peut imaginer ce mécanise n'est utile que pour des mécanismes cognitifs très primitifs. Par exemple, cette architecture est utilisée pour des robots uni-tâche tel que des robots utilisés dans les usines de fabrication.

L'autre l'architecture étant entièrement délibérative permet de concevoir de mécanismes cognitifs évolués mais donne une réponse plus lente (Figure 2.8). Par exemple, cette architecture est utilisé comme un mécanisme de décision pour des robots qui essai de créer une carte géographique.

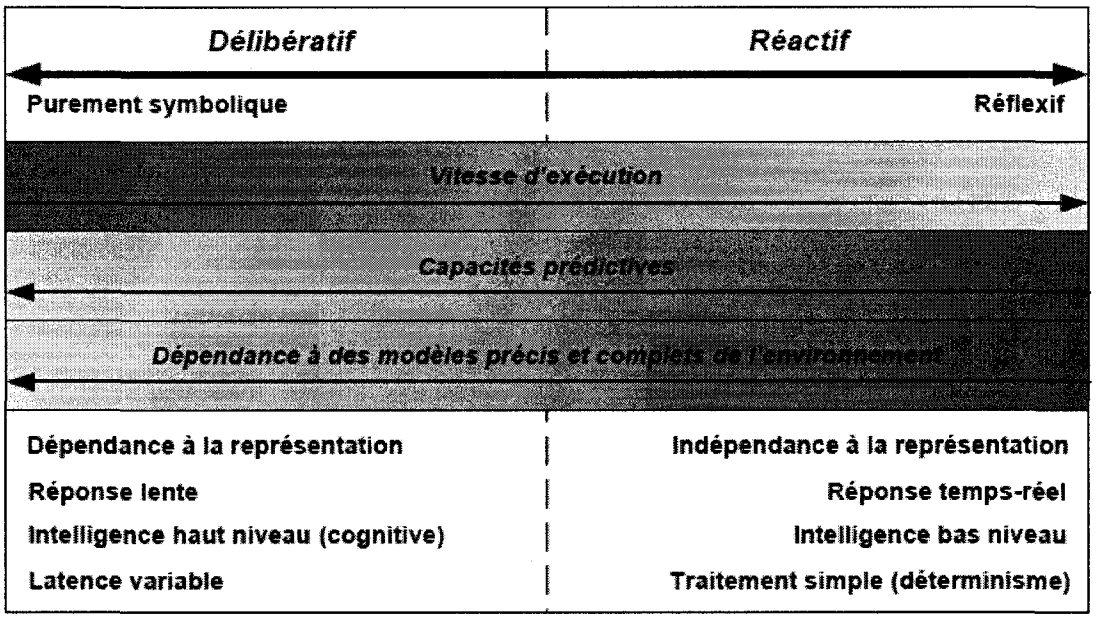


Figure 2.8 : Architecture délibératif versus réactif (Tiré de J. Beaudry)

Une architecture décisionnelle hybride munie de capacités délibératives et réactives est utilisée pour les robots utilisés dans le cade ce travail de maîtrise. En utilisant une architecture multicouche, les capacités réactives sont développées en tant que des comportements de bas niveau qui vont permettre

de réagir sur l'environnement en commandant les actuateurs. Les capacités délibératives sont alors situées à des couches supérieures à la couche réactive et permettent une intelligence de plus haut niveau. En variant le nombre de couches de la partie délibérative, on peut moduler les capacités réactives et délibératives. La figure suivante présente ce modèle d'architecture, qui peut être vu comme un modèle traditionnel à trois couches.

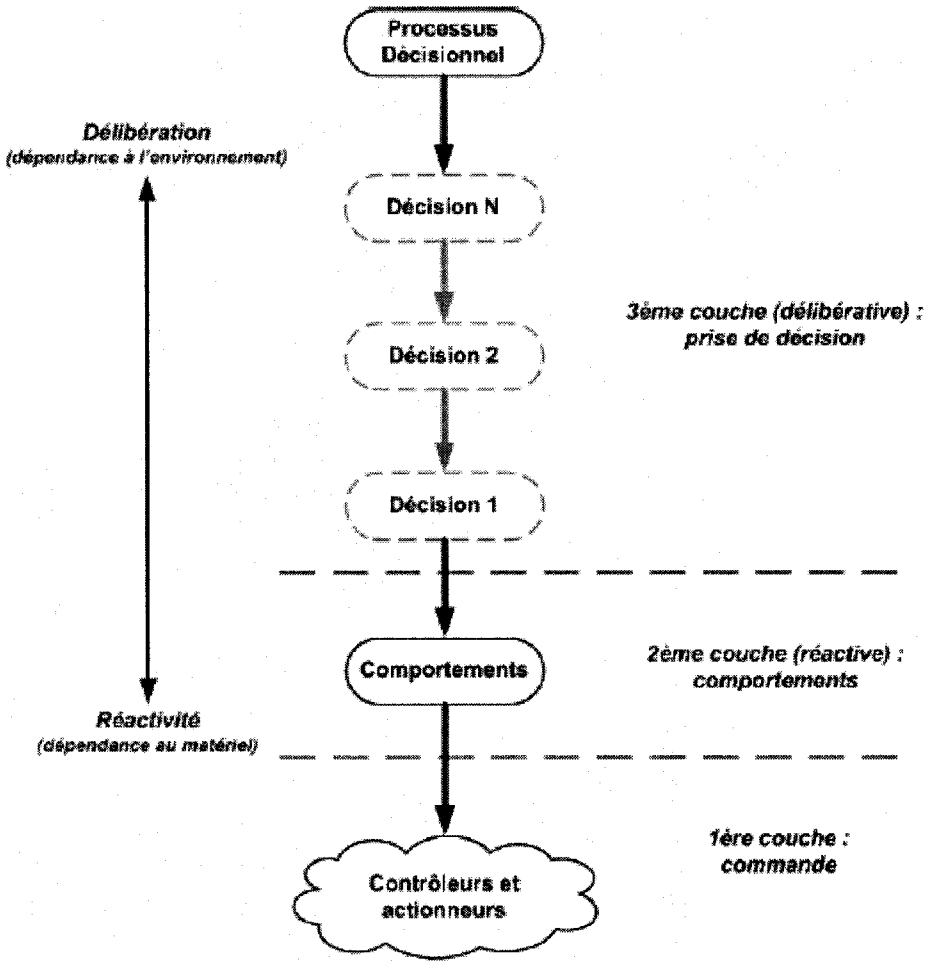


Figure 2.9: Mécanisme décisionnel à trois étages (Tiré de J. Beaudry)

Il est évident qu'un processus décisionnel constitué de plusieurs niveaux hiérarchiques présentera une planification plus augmentée par rapport à une prise de décision basée sur un seul étage (strictement réactive). Cette architecture a été intitulée la machine décisionnelle hiérarchique (HDM) par J. Beaudry.

2.2.12. *Platform de simulation*

Une plateforme en simulation a été développée en grande partie en utilisant la librairie *MICROB*. En fait, au niveau du logiciel de contrôle, seulement la classe *Controller* qui sert à d'accéder à la carte de contrôle a été modifié en utilisant les modèles cinématique et dynamique de la plate-forme à vitesse différentielle (DeSantis) ou omnidirectionnelle. Aussi un logiciel pour simuler la dynamique du ballon ainsi qu'un visualisateur 3D complètent ce simulateur.

Ce simulateur a été grandement utilisé et modifié dans le cadre des travaux de maîtrise et afin de développer les différents comportements des robots, car il est alors possible de faire des essais sans utiliser le système réel. L'architecture du système de l'apprentissage machine a été d'abord implémenté et testé sur ce simulateur. La figure suivante donne un aperçu de ce module de simulation avec une interface (situé à droite) qui permet de contrôler et vérifier l'état des robots.

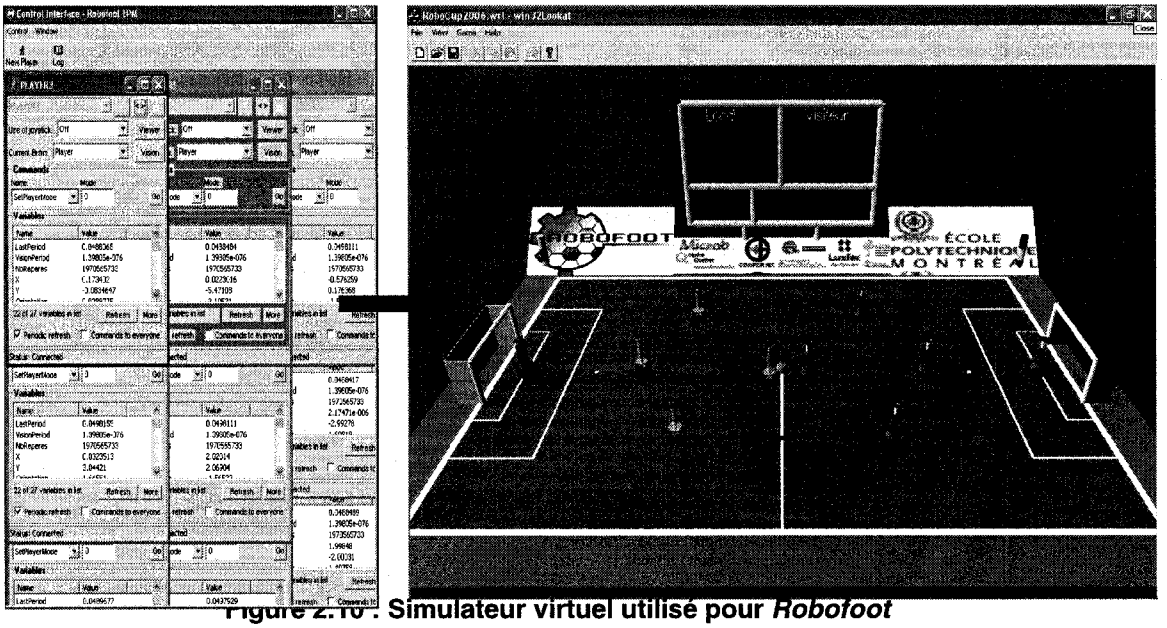


Figure 2.10 : Simulateur virtuel utilisé pour *Robofoot*

Chapitre 3
Implémentation du système d'apprentissage sur un système multi-robot

3.1. Système d'apprentissage choisi

Pour mettre au point un système d'apprentissage pour un système multi robot, il a été établi que c'est quasiment impossible d'effectuer un apprentissage qui relit directement les capteurs aux actionneurs dans le cas des robots joueurs de soccer (P. Stone. 2000) puisque le temps d'apprentissage sera trop élevé pour que ceci soit efficace. Une approche qui divisera le problème d'apprentissage en sous parties sera donc entreprise. On tentera alors d'apprendre à partir des comportements bas niveau et les combiner pour progresser vers les comportements haut niveau. Utilisant cette ***approche d'apprentissage multi niveaux (hiérarchisé)***, différents algorithmes d'apprentissage peuvent être utilisés pour chaque niveau. Le résultat d'apprentissage de chaque niveau est donc utilisé par un niveau supérieur.

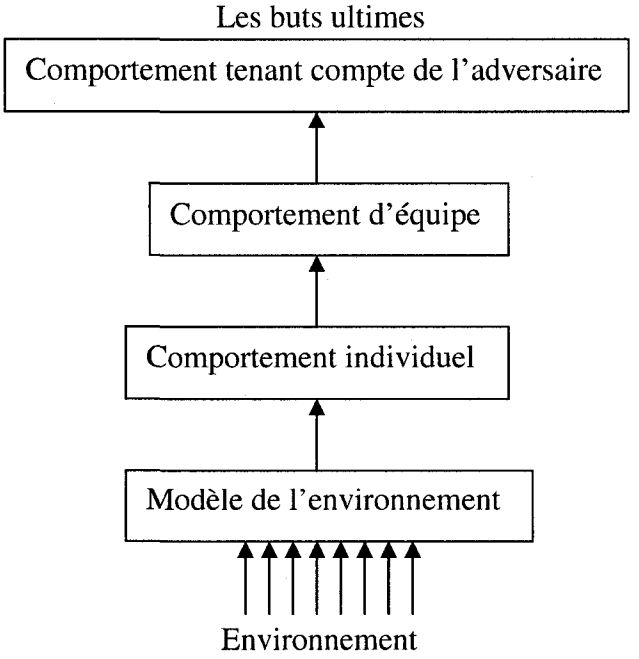


Figure 3.1 : Approche d'apprentissage multi niveaux (hiérarchisé)

À chaque niveau, un apprentissage a lieu de façon indépendante et chaque niveau fourni les entrées du niveau supérieur, sauf dans le cas du premier niveau ou les entrées ne proviennent que de l'environnement.

Il est aussi important de noter qu'afin de régler le problème d'apprentissage pour un niveau, il y a un compromis à faire entre l'exploitation de la solution optimale trouvée et l'exploration des nouvelles possibilités (en utilisant de façon non optimale les informations qui nous sont fourni par apprentissage ou en utilisant le hasard pour laisser de la place pour de nouveaux apprentissages).

3.2. Librairie utilisé pour faire l'apprentissage (Torch3)

La librairie Torch3 est une librairie qui permet d'avoir une base pour faire certains types d'apprentissage. En fait, elle fournit une fonction de décision f(x) convenable qui prédit la valeur de sortie, en lui fournissant une série d'exemples d'entraînement.

Torch est une librairie modulaire écrit en C++ sous une licence BSD (une licence libre utilisée pour la distribution de logiciels qui permet de réutiliser tout ou partie du logiciel sans restriction). Il existe seulement quatre classes principales dans cette librairie qui ont été fortement utilisé dans le cadre de ce travail de maîtrise, soit :

- La classe ***DataSet*** : servant à gérer les données et les emmagasiner en mémoire ou sur le disque dur.
- La classe ***Machine*** : qui prend des entrées et certains paramètres et retourne une sortie en se basant sur un modèle déjà conçu par la classe *Traniner.*
- La classe ***Trainer*** : entraîne et teste une *Machine* avec un certain *DataSet*
- La classe ***Measurer*** : cette classe permet de mesurer l'erreur du modèle sous différents formats standards tel que l'erreur moyenne au carré (MSE).

Le fonctionnement de base de cette librairie est relativement simple. D'abord, on emmagasine des exemples sous forme d'un ***DataSet***. Ces données sont ensuite utilisées par le ***Trainer*** pour bâtir un modèle qui sera utilisé par la classe ***Machine*** pour calculer la sortie à partir d'une entrée et certains paramètres donnés. Finalement, on peut utiliser la classe ***Measurer*** pour quantifier l'erreur associée au modèle.

3.3. Sujets d'apprentissage

Les travaux de recherche de ce projet vont porter sur l'apprentissage de certains éléments importants d'un système multi-robot joueurs de soccer en commençant par des éléments de bas niveau et propre à un agent et en se dirigeant vers des éléments de plus haut niveau tel que la formation et le parton de jeu choisi dans un match.

- Les robots joueurs de soccer doivent commencer par apprendre à manipuler le ballon avant de pouvoir s'intégrer dans une équipe. Alors, le premier sujet, qui constitue aussi la couche la plus élémentaire de cette architecture, sera l'étude de l'apprentissage de l'interception du ballon, du tir vers le but ou vers une cible et de la protection du but par le gardien.

- En deuxième lieu, il est pertinent d'étudier l'interaction la plus élémentaire entre les robots : une passe entre les robots en se servant des résultats obtenus de l'interception du ballon et des tirs vers une cible.

- Le choix entre différents patrons de jeu sera étudié pour des situations données en se basant sur la réussite de chaque patron : une fonction pondérée des paramètres suivants:

    - Différentiel de buts (marqués – contres)
    - Temps de possession de ballon
    - Pourcentage de temps joué dans la zone adverse

  Chaque patron sera constitué d'une ou plusieurs formations et se servira donc de l'apprentissage de la couche précédente.

3.4. Apprentissage d'un comportement individuel

3.4.1. Apprendre le comportement d'interception du ballon (Catch ball)

L'interception du ballon est très importante pour les défenseurs, et de façon similaire très importante pour capter des passes dans le cas d'un attaquant. Pour munir les agents de cette technique, on peut soit prédire la position future du ballon en estimant la vitesse de celui-ci, ou on peut recueillir des données sur les interceptions réussies et créer un comportement général de l'interception de ballon de façon empirique. Pour les fins de ce mémoire, la deuxième méthode est choisie.

La méthode actuellement utilisée par les joueurs (sauf le gardien[2]) pour intercepter le ballon est de diriger le robot directement vers le ballon par défaut et s'il est du mauvais coté du ballon il revient à l'arrière du ballon. Mais cette approche n'est performante que dans le cas où le ballon est stationnaire. Dans le cas où le ballon suit une trajectoire quelconque, une bonne approche sera celle qui prend en considération la direction et la vitesse du ballon et qui force le robot à intercepter le ballon en conséquence. Pour apprendre ce comportement, on peut s'inspirer d'une approche utilisée par P. Stone (2003), et utiliser l'algorithme suivant :

TANT QUE Dist_Ball > d_0,
{
    Tourner(Angle_Ball)
}

SI Dist_Ball < d_0 ALORS
{
- Angle_R = une angle au hasard entre -45° et 45 °
- Foncer vers l'avant avec une angle de Angle_Ball + Angle_R
- Recueillir les informations suivantes : Vitesse Ballon, Angle_Ball et Angle_R
- Déterminer l'échec ou le succès de l'essai et sauver le résultat.

}

[2] Le gardien calcule la position du ballon lorsqu'il va arriver sur la ligne du but et il se met à cette position.

Où la vitesse du ballon peut être calculée en utilisant la position du ballon aux instant t et t1 . Alors, le robot commence par tourner face au ballon en faisant une rotation mais sans changer de position en x et y. Il continuera ainsi jusqu'à ce que le ballon atteigne une certaine distance d_0, il va ensuite choisir un angle quelconque *(Angle_R)* et finalement avance dans cette direction *(Angle_R + Angle_Ball)*. On tente ici d'apprendre la bonne valeur pour cet angle *(Angle_R)*. En fait, non seulement le robot va apprendre ici la bonne direction à choisir en fonction de la direction et la vitesse du ballon (ce qui se fait simplement par une équation mathématique) mais aussi il va compenser pour toutes sortes de perturbation inconnue ou d'erreur introduit dans le système due aux différents facteurs tel que la friction du terrain qui est toujours variante ou les erreurs d'estimation de position et de vitesse calculées par le système de vision

3.4.2. L'architecture de test

On positionne le robot devant le but et, on tente de faire un but à travers la zone surveillée par le robot. Le ballon est lancé à partir des différents endroits. Une réussite s'agit de faire dévier le ballon de sorte que le ballon rebondit sur lui et change de direction pour éviter un but. Tel que mentionné précédemment, le robot suit le ballon en changeant son orientation pour faire toujours face au ballon jusqu'à ce que le ballon arrive à 1 mètre de lui. Puis, il change son orientation d'un angle choisi de façon aléatoire lors des premiers essais et déterminé par apprentissage par la suite, puis avance dans cette direction.

En simulation, une estimation de la vitesse du ballon et de sa direction a été calculée en utilisant la position actuelle du ballon et celle de l'instant précédant.

Un script a été écrit pour automatiser les tirs au but en utilisant le *SUI*[3] offert par les librairies de *Microb*. Le *SUI* est une interface en ligne de command qui permet de communiquer avec un serveur défini par les librairies de *Microb*.

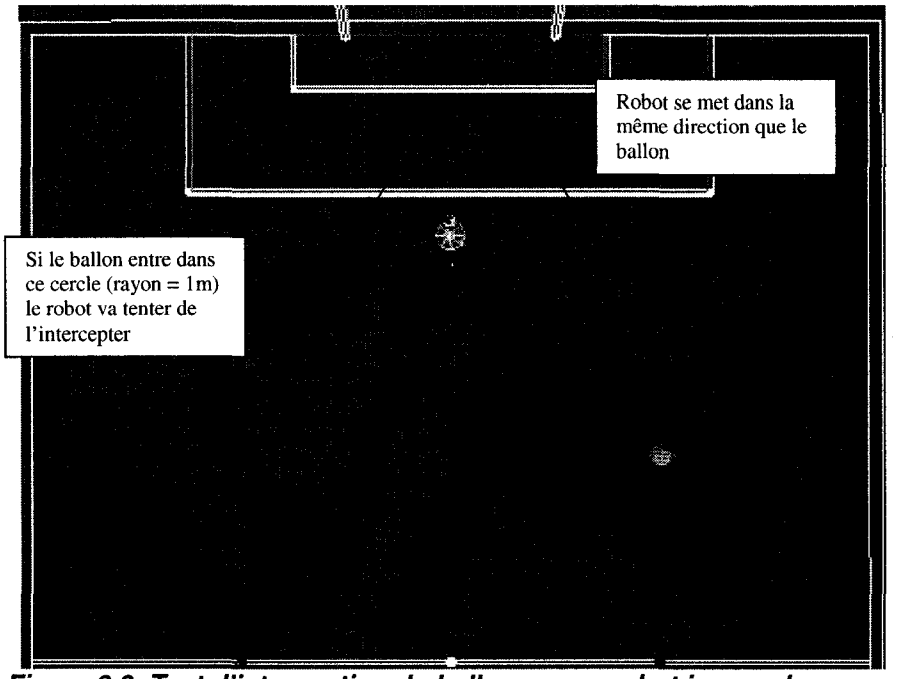


Figure 3.2: Test d'interception du ballon par un robot joueur de soccer

Le script créé fait appel au service défini pour le *soccerfield_server* qui permet de simuler les comportements du ballon. On peut alors simuler un tir au but soit en appelant la fonction service *botter_ballon(position x, position y, vitesse, angle)*, ou son abréviation bb dans ce cas. Le script (*MyScript*) utilise quelques milliers de commandes d'entraînement. La figure suivante illustre ce mécanisme:

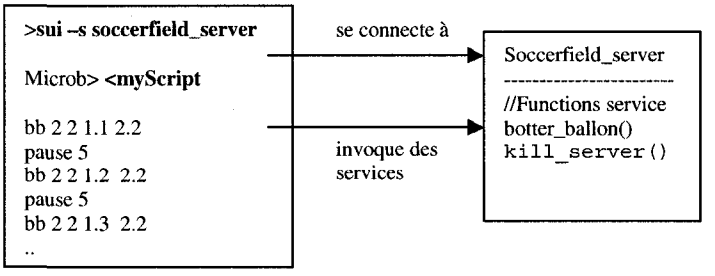


Figure 3.3 : Mécanisme de communication entre le SUI et le Soccerfiel_server

[3] Server User Interface : une interface en ligne de command qui permet de communiquer avec un serveur défini par les librairies de *Microb*

Le *soccerfield_server* qui fait partie de *soccerfield* simulator se connecte à son tour au serveur *team_server* qui communique les changements à tous ses clients (tous les *Player* et *Lookat 3D*).

Sachant que nous avons les vecteurs d'entrée et de sortie suivants :

X = [Vitesse_ballon, Direction_ballon, Angle_R], Y = [Angle_final_R]

Un nouveau comportement a été ajouté au robot joueur de soccer : « `CatchBallLearning` » qui implémente l'algorithme de l'interception du ballon en sauvant dans un fichier les valeurs de *X* et *Y* du cas où l'interception a été une réussite. Une réussite dans l'interception du ballon correspond à faire dévier la direction du ballon vers le but adverse afin de s'assurer que le ballon ne se trouve par entre le robot et le but (ou tout simplement si le ballon n'entre pas dans le but). Voici l'affichage en console pour ce comportement :

Figure 3.4 : Création du fichier de données d'entraînement pour créer le modèle d'apprentissage SVM par régression

Ceci fournit un fichier contenant des données d'entraînement pour notre modèle à apprentissage *SVM* par régression. Une fois le modèle créé grâce au nouveau module *RobofootTorch* basé sur la librairie *Torch3*, on peut prédire la valeur de la direction (*Angle_Ball + Angle_R*) que le robot doit prendre en fonction d'une vitesse et d'une direction donnée du ballon ainsi que l'angle que fait le ballon par rapport au robot (*Angle_R*), tel qu'illustré à la figure 3.4.

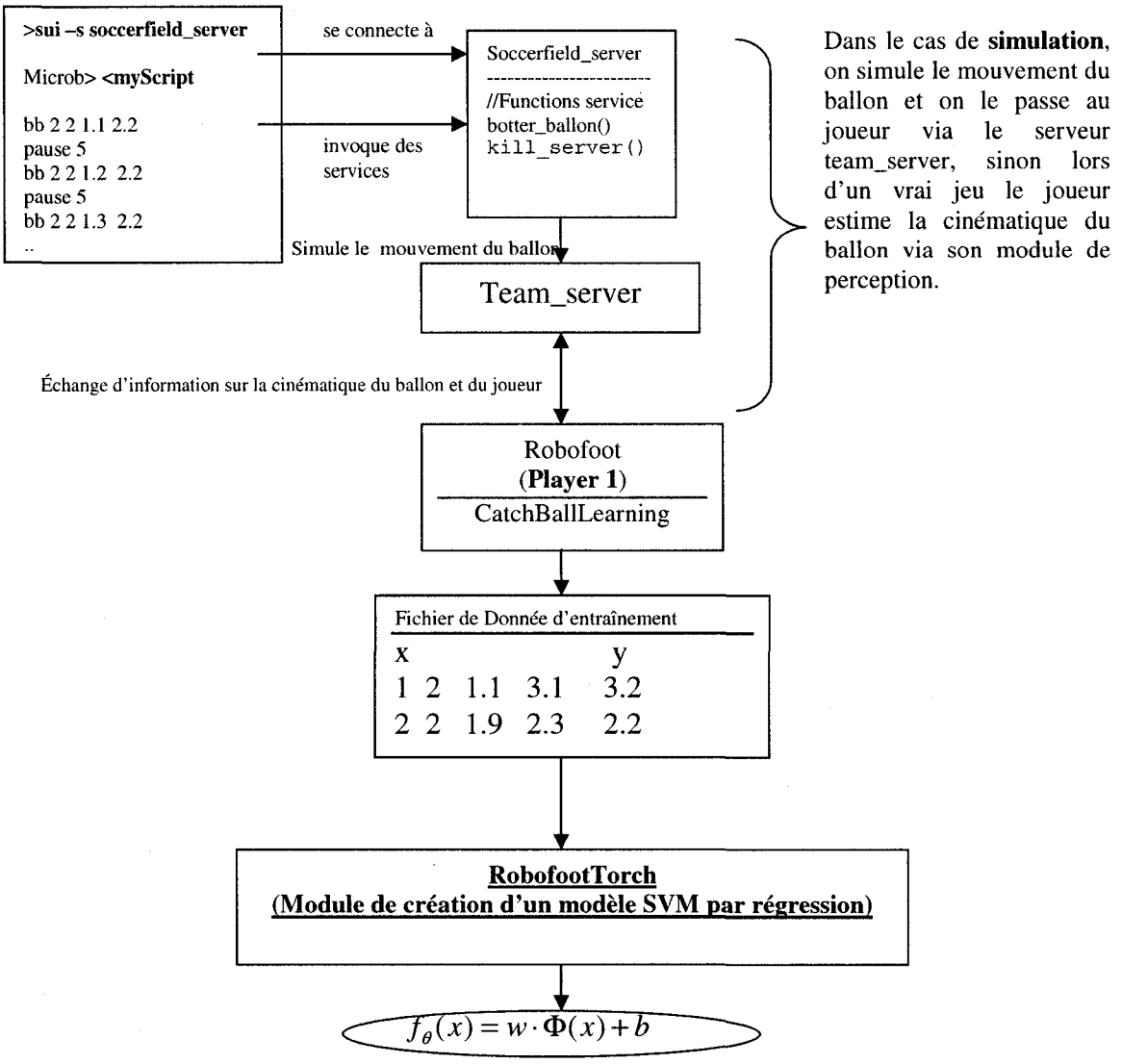


Figure 3.5 : Mécanisme d'entraînement du modèle d'apprentissage de la réception de ballon par SVM

Ceci complète une séance d'entraînement permettant *RobofootTorch* à créer un fichier décrivant la fonction f(x).

La prochaine étape consiste à tester notre modèle en définissant un nouveau comportement pour un robot soit le « `CatchBallLearned` ». Ce comportement suit le même principe que celui de « `CatchBallLearning` »

mais dans ce cas au lieu de donner une valeur aléatoire pour l'angle Angle_R, on établit la valeur de la direction (vers la quelle le robot doit avancer pour garantir une interception) à partir du modèle établi précédemment par *RobofootTorch*. En simulation, ceci a été rendu possible en ajoutant le module de *RobofootTorch* comme un serveur dans l'architecture client/serveur du système multi robots. Chaque robot fait donc une requête auprès du serveur qui à son tour trouve le bon angle utilisant le modèle bâti par apprentissage et renvoi cette valeur au robot.

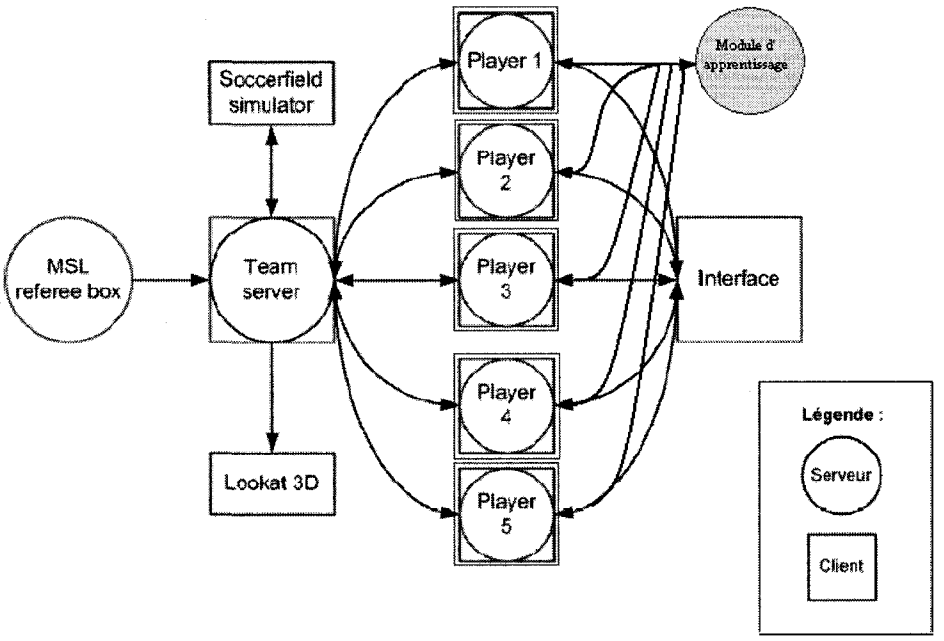


Figure 3.6: Schéma de l'architecture Client/serveur du système multi robots avec l'ajout du serveur du module d'apprentissage et son interaction dans le système (modifié de J. Beaudry)

Mais après avoir fait plusieurs tests expérimentaux, le module d'apprentissage a été directement incorporé dans le module *Player* pour éliminer le temps d'attente causé par le délai de communication client-serveur.

Le modèle en fournissant à *RobofootTorch* une série d'exemples test (où la valeur de y est connu) pour trouver l'erreur associé à ce modèle. En ajustant la valeur des paramètres de notre modèle, la meilleur combinaison a été trouvée (celle qui minimise l'écart entre la valeur vrai valeur de y et celle trouvée par *RobofootTorch*).

Ensuite, en se servant de ce modèle lors de la situation de jeu, dans le cas où l'interception est réussie on peut décider aussi d'ajouter cet ensemble d'entrée/sortie à notre fichier d'entraînement pour qu'il puisse s'en servir pour bâtir un modèle plus robuste lors du prochain entraînement, alors à chaque fois on doit rebâtir le modèle, ce qui consiste à refaire la lecture du fichier d'entrainement. Il faut donc s'assurer que le fichier d'entraînement ne devient pas trop volumineux, ce qui augmentera aussi la taille du fichier modèle, et qui causera alors un ralentissement dans le processus de décision du robot pour trouver l'angle approprié. Afin de remédier à ce problème, les entrées du modèle sont discrétisées et une étude a été faite sur l'effet de la discrétisation des entrées sur le taux de succès des interceptions.

La figure suivante présente un aperçu graphique d'une séance d'entraînement dans le cas de simulation. La fenêtre 1 montre le script utilisé pour faire des tirs au but avec des vitesses différentes et des angles différents. La fenêtre 2 nous permet de visualiser les valeurs obtenues pour une interception réussie.

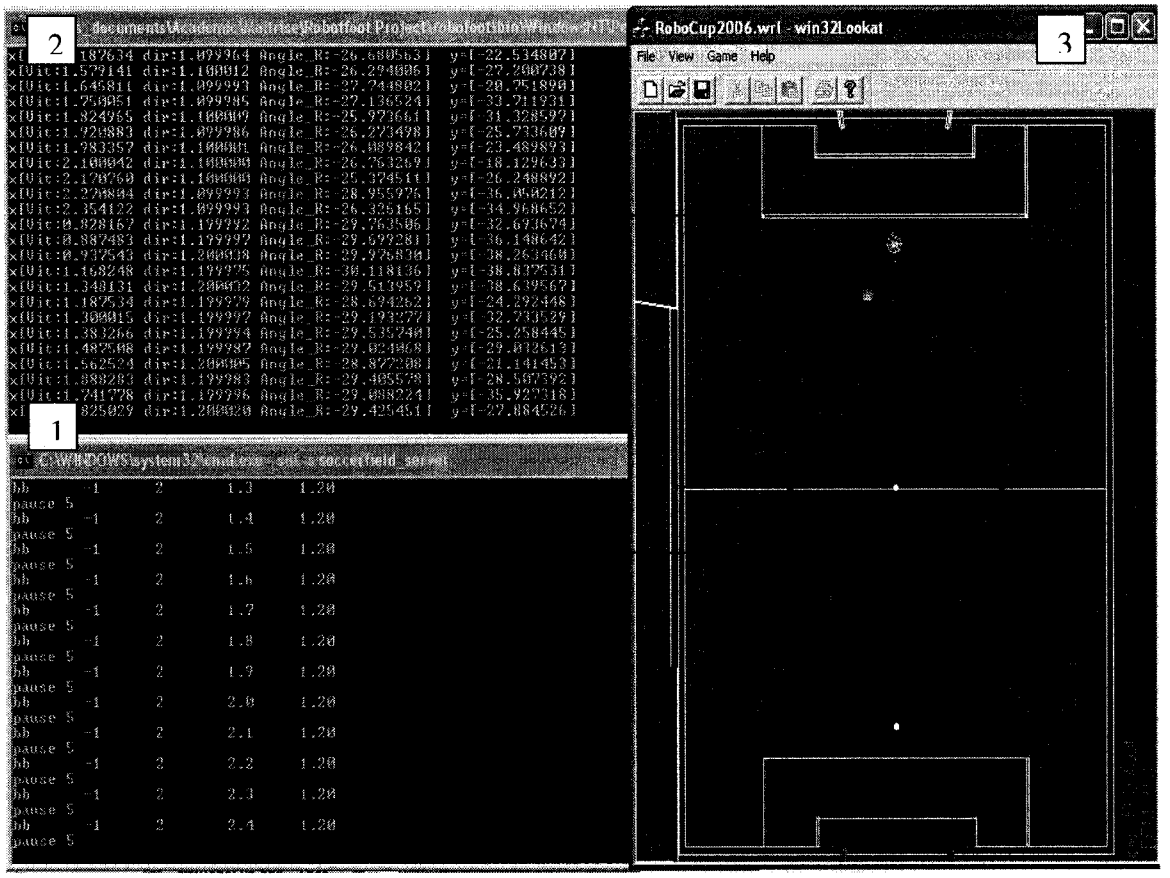


Figure 3.7 : Exemple graphique d'une séance d'entraînement

3.5. Les résultats pour la méthode d'apprentissage par SVM

3.5.1. Simulation

En simulation, les résultats obtenus ont été validés de deux manières différentes. D'abord, le même fichier qui avait été utilisé pour bâtir le modèle (la relation entre la sortie *Y* et les entrée *X*) a été aussi utilisé pour estimer l'angle que le robot doit prendre pour intercepter le ballon, ce qui n'est pas nécessairement la meilleur approche mais une approche qui nous permet en peu de temps de trouver la meilleur combinaison pour les valeur des paramètres de notre modèle.

L'erreur au carrée moyenne (*MSE*) a été calculée soit la différence entre le *Y* trouvé en utilisant le modèle bâti par *RobofootTorch* et celui trouvé de façon expérimentale dans le fichier de donné d'entraînement. Le module *RobofootTorch* crée un modèle SVM. Rappelons nous que l'on essai de créer le modèle de forme suivante :

$$f_\theta(x) = w \cdot \Phi(x) + b = b + \sum_{l=1}^{L} \alpha_l y_l \Phi(x_l) \cdot \Phi(x) = b + \sum_{l=1}^{L} \alpha_l y_l k(x_l, x) \qquad (3.1)$$

Afin de prédire une sortie *f(x)* en se basant sur la valeur voulue pour les paramètres suivants :

- La valeur de σ pour le *Kernel* Gaussien qui se trouve dans l'équation suivante :

$$k(x_1, x_2) = \exp(-\|x_1 - x_2\|^2 / (2\sigma^2)), \qquad (3.2)$$

- La valeur du facteur de compromis *C* (qui remplace 1/ *L* et μ dans l'équation 3.3) dans l'équation suivante :

$$\frac{1}{2}\|w\|^2 + C\sum_{l=1}^{L}(\xi_l + \xi_l^*) \qquad (3.3)$$

- La valeur de l'epsilon (ε) à minimiser dans l'équation suivante :

$$\forall l \quad \begin{matrix} (\omega \cdot \Phi(x_l) + b) - y_l \leq \varepsilon + \xi_l \\ y_l - (\omega \cdot \Phi(x_l) + b) \leq \varepsilon + \xi_l^*, \end{matrix} \tag{3.4}$$

- Nombre de donnée d'entraînement

Voici les résultats obtenus :

MSE (degré^2)	Erreur en degré	Σ	C	e	Nombre de données d'entraînement
65.11	8.07	100	100	0.7	700
65.06	8.07	100	100	0.7	1000
62.93	7.93	100	200	0.7	1000
42.61	6.53	1	100	0.7	700
42.91	6.55	1	100	0.7	1000
464.04	21.54	1	10	0.7	1000
31.77	5.64	1	200	0.7	1000
11.99	3.46	0.01	100	0.7	700
11.05	3.32	0.01	100	0.7	1000
0.01	0.10	0.01	200	0.7	1000
11.99	3.46	0.01	100	1	1000
11.06	3.33	0.01	100	0.1	1000
11.05	3.32	0.01	100	0.01	1000
11.52	3.39	0.0001	100	0.7	1000

Tableau 3.1 – L'erreur trouvée en fonction de paramètre du modèle trouvé par un apprentissage basé sur la méthode de SVM par régression

On remarque facilement que la valeur de σ est directement proportionnelle à la valeur de l'erreur entre l'angle trouvé et le bon angle (qui assure l'interception). Aussi, on trouve une relation inversement proportionnelle entre les valeurs du compromis C est l'erreur trouvé. Une diminution de la valeur de l'epsilon peut aussi diminuer l'erreur mais de façon très peu significative comparée à l'effet de C et σ. En plus, lorsque l'on augmente le nombre de données d'entraînements, l'erreur diminue, mais si l'on augmente beaucoup le nombre d'exemples d'entraînement, la prise de décision pour trouver le bon angle devient plus lent. Donc, il faut faire un compromis en termes de

nombre d'exemples d'entraînement entre la vitesse de prise de décision et son exactitude.

Il faut tenir compte du fait que les angles trouvés de façon expérimentale ne représentent pas les meilleures solutions pour une situation donnée mais seulement une solution possible parmi plusieurs.

Une autre manière de tester les résultats obtenus est de refaire les mêmes essais en simulation utilisant le script utilisé lors de la séance d'entraînement.

Dans ce cas, le taux de réussite a été calculé, soit le nombre de tirs interceptés (essais réussis) divisé par le nombre de tir vers le robot. En moyenne, il y a eu une amélioration de plus de 25% versus le taux de réussite utilisant l'angle du robot additionnée par un angle au hasard, soit une augmentation de 25% à plus de 53% de réussite ce qui est équivalent d'intercepter un tir sur deux. Aussi, il faut noter que dans la moitié des cas de tir la vitesse du ballon est supérieure à la vitesse du robot (les tirs ont une vitesse comprise entre 1m/s à 3 m/s et la vitesse du robot est de 2m/s), et donc une augmentation dans la vitesse du robot engendrera sûrement une amélioration dans le taux de réussite des interceptions.

Pour ce qui concerne l'effet du bruit, pour simuler une incertitude dans le cas des vrais robots, un bruit de .1 m/s a été ajouté sur l'estimation de la vitesse et 5 degré pour l'estimation de la direction du ballon, et ceci a diminué le rendement de ce comportement de 4%.

3.5.1.1. Discrétisation

Tel que mentionné dans les sections précédentes, afin de diminuer la taille de nos fichiers d'entraînement et notre modèle, les entrées x de notre système sont

discrétisés. Le tableau suivant nous permet de comparer les résultats obtenus utilisant différentes valeurs pour arrondir les entrées de notre système.

Δx_1 (m/s)	Δx_2 (degré)	Δx_3 (degré)	L initiale	L discrétisation	taux de réussite (pourcentage)
0.1	2	2	725	704	56.6%
0.1	3	3	725	675	56.1%
0.1	5	5	725	567	56.2%
0.2	2	2	725	603	50.9%
0.3	3	3	725	491	51.6%
0.4	4	4	725	354	50.7%
0.5	5	5	725	273	49.4%

Tableau 3.2 – Taux de réussite de l'interception du ballon en fonction des valeurs utilisées pour arrondir les entrées

On remarque facilement qu'une discrétisation sur la valeur de x_1 (soit la vitesse du ballon) affecte beaucoup plus le taux de réussite qu'un arrondissement en degré des angles. Alors, des approximations de 0.1 m/s (pour le x_1) et 5 degrés (pour le x_2 et x_3) ont été choisi pour discrétiser le modèle.

Il est important de mentionner que sans discrétisation le modèle créé devenait trop complexe (sans pourtant devenir plus précis) et son temps d'exécution devenait trop élevé pour un tel système qui doit réagir en temps réel.

3.5.2. Résultats expérimentaux

Les premiers tests sur le robot ont été effectués en utilisant la même architecture client-serveur qu'en simulation. Le robot était le client et un ordinateur portatif (2GHz de CPU et 1GB de RAM) le serveur (voir figure 3.8). Les mêmes fichiers modèles trouvés en simulation ont utilisés servis pour ces premiers tests.

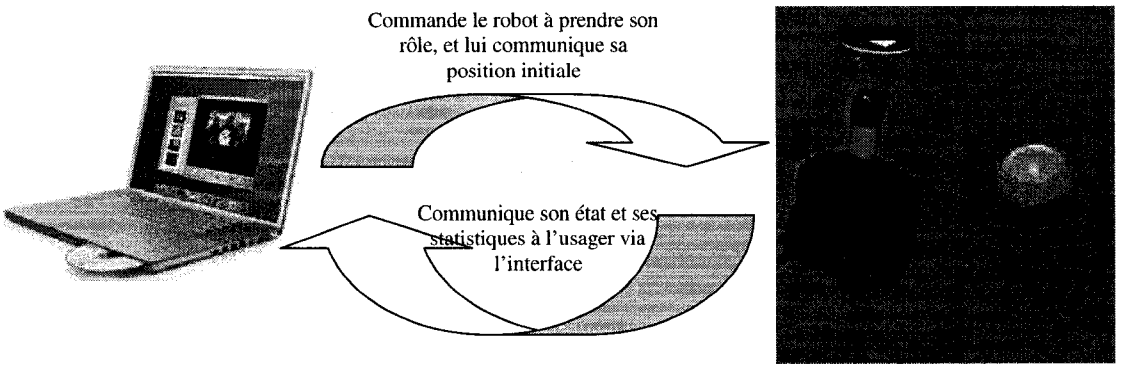


Figure 3.8 : Communication entre l'ordinateur portable et le robot

Malgré le taux élevé de réussite obtenu en simulation, ces premiers tests ont donné un taux de réussite d'environ 15%. Après investigation les différents facteurs causant cette diminution de performance ont été identifiés. Il s'agissait d'abord d'un délai de communication entre le robot et le serveur, ce qui empêchait le robot de réagir assez rapidement. Aussi, les imprécisions dues à l'estimation de la vitesse et la direction du robot par le système de vision ont contribué à cette diminution de performance.

Le module d'apprentissage a été donc incorporé au module *Player* permettant d'augmenter le taux de réussite du robot à quasiment le même taux qu'obtenu en simulation. Par contre, à cause des différents éléments qui causaient un délai, le rayon du cercle utilisé pour débuter le comportement d'interception a été augmenté de 15cm.

Le taux de réussite final de 45% a été trouvé sur 40 tirs, mais il faut mentionner que vue l'impossibilité de refaire exactement les mêmes tirs et aussi à cause des imprécisions du système de vision ce taux peut varier (et descendre parfois jusqu'à 25%).

3.5.3. Résultat pour la méthode d'apprentissage par MLP

Tout comme dans le cas d'apprentissage par SVM, les résultats obtenus ont été validés encore de deux manières différentes. D'abord, le même fichier qui avait été utilisé pour bâtir le modèle (la relation entre la sortie Y et les entrée X) a encore été utilisé pour estimer l'angle que le robot doit prendre pour intercepter le ballon. L'erreur au carrée moyenne (MSE) a été calculée soit la différence entre le Y trouvé en utilisant le modèle bâti par *RobofootTorch* et celui trouvé de façon expérimentale dans le fichier de donné d'entraînement. Rappelons-nous que le module *RobofootTorch* crée un modèle MLP basé sur l'équation suivante :

$$f_\theta(x) = w \cdot \Phi(x) + b \tag{3.5}$$

avec

$$\Phi(.) = (\Phi_1(.), \Phi_2(.), ..., \Phi_n(.)) \tag{3.6}$$

qui représente le nombre de neurones sur la couche cachés. L'erreur trouvée en degré s'agit de la différence entre l'angle sauvegardé lors des séances de l'entrainement pour une certaine entrée et l'angle trouvé par le modèle MLP utilisant le même vecteur d'entrée. Voici les résultats obtenus pour divers nombre de données d'entrainement et d'unités cachées :

MSE	Erreur en degré	Nombre d'unités cachées	Nombre de données d'entraînements
33508	183.1	10	300
5100	71.4	80	700
600	24.5	40	700
595	24.4	11	700
457	21.4	10	700
514	22.7	9	700
957	30.9	8	700
847	29.1	5	700
480	21.9	20	2055
460	21.4	11	2055
304	**17.4**	**10**	**2055**
370	19.2	9	2055
450	21.2	5	2055

Tableau 4.3 – L'erreur trouvée en fonction de paramètre du modèle trouvé par un apprentissage basé sur la méthode de MLP par régression

En analysant les valeurs obtenues, on remarque qu'en augmentant le nombre de données d'entraînement, l'erreur en degré diminue de façon beaucoup plus prononcée que dans le cas d'apprentissage par SVM. On remarque aussi que pour un nombre d'unités cachées de 10, on obtient le moins d'erreur.

Il est important de mentionner ici la théorie de la régularisation proposé par Tikhonov qui permet d'optimiser l'apprentissage dans un MLP. La technique de régularisation consiste à imposer des contraintes, donc à apporter une information supplémentaire, sur l'évolution des poids du réseau de neurones. Utilisant la formule suivante dans laquelle on doit trouver la bonne valeur de γ :

$$R(w) = Q_{Erruer} + \gamma * Q_{Poids} \tag{3.7}$$

Où Q_{Erruer} est le terme utilisé afin de mesurer la performance. Par exemple (tel que décrit par l'équation 1.10) l'erreur entre la valeur vrai valeur et la valeur prédite par MLP au carrée et Q_{Poids} est le terme qui calcule la pénalité associée à la complexité de

(3.8)

MLP, on peut par exemple pénaliser les poids trop grands selon la formule suivante (Frédéric Szczap 2006):

$$Q_{Poids} = \frac{1}{N}\sum w_i^2$$

Quoiqu'utile, dans le cadre des travaux de ce maitrise, l'effet de la régularisation n'a pas été étudié (γ a toujours pris la valeur de 0).

En faisant une comparaison des résultats obtenus pour la méthode d'apprentissage par SVM, on peut alors conclure que la méthode d'apprentissage par régression de la méthode d'apprentissage par MLP donne des performances de beaucoup inférieures à celle d'apprentissage par SVM. Dans le cas de MLP, nous ne pouvons bâtir un modèle exact que si nous avons un grand nombre de données d'entraînements. Mais sachant que un entraînement avec plus de 2500 données dans notre cas engendre un modèle très lourd à manipuler et le processus de décision devient trop lent pour être acceptable en temps réel, il devient évident que le modèle bâti par la méthode SVM est plus approprié dans ce cas.

Une autre manière de tester les résultats obtenus tout comme dans le cas d'apprentissage par la méthode SVM a été explorée, soit de refaire les mêmes essais en simulation utilisant le script utilisé afin de trouver les données d'entraînement.

Le taux de réussite a été calculé, sois le nombre de tirs interceptés (essais réussis) divisé par le nombre de tir vers le robot. Un taux de réussite de 43% a été obtenu utilisant dix unités cachées et 2055 données d'entraînement. Ceci est en effet 10% inférieur aux taux trouvé en moyenne dans le cas de SVM.

3.6. Apprendre à effectuer un tir

L'opération de tir vers le but constitue un élément important parmi les opérations de base d'un robot joueur de soccer.

3.6.1. Définition d'un tir dans le cas d'une passe

D'abord, il faut préciser ce qui constitue un tir approprié dans le cadre de ce travail. On peut définir un tir approprié comme étant un tir qui est effectué pour arriver avec une vitesse donnée dans une zone délimité par un demi-cercle d'un rayon défini qui se trouve devant le receveur. La vitesse choisie est entre 0.2 et 0.6 m/s dans le cas dont le tir est ciblé vers un coéquipier et le ballon doit arriver dans un cercle de rayon de 0.2m devant le robot.

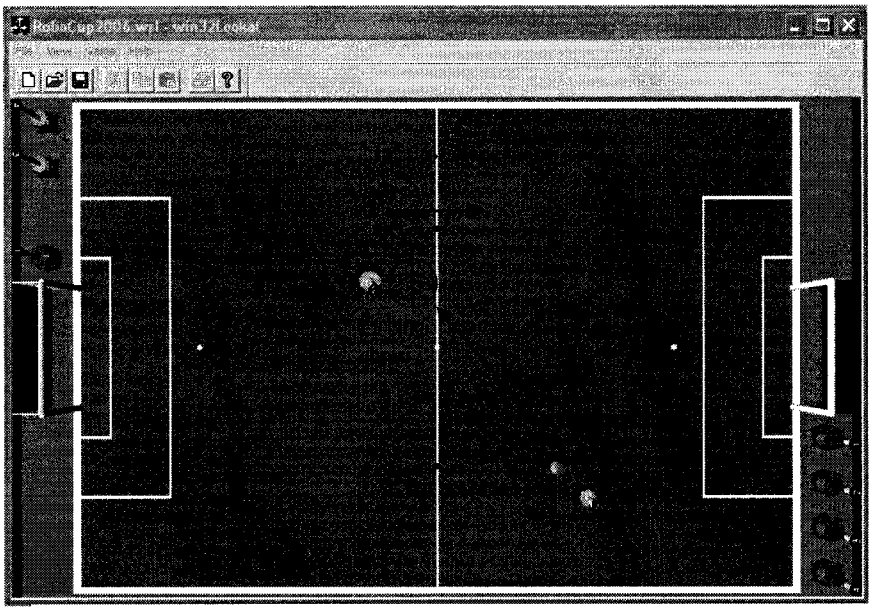


Figure 3.9 : Un tir de passe appropriée

3.6.2. Scénario d'entraînement

Tout comme dans les cas précédents, un entraînement pour recueillir des données pour pouvoir bâtir un modèle est nécessaire. Le scénario choisi pour faire les tests (l'entraînement du notre modèle) consiste à faire une tentative de tir vers une cible donnée, le robot va donc se mettre derrière le ballon à l'intersection d'un cercle (dont le centre est la position du ballon et qui a un rayon relatif à la distance entre le ballon et la cible) et la ligne droite qui rejoint le ballon à la cible. Ensuite, il avancera vers le ballon et faire son tir avec une intensité aléatoire. La valeur de l'intensité de tir est dans ce cas l'objet de notre apprentissage. En simulation le robot va ensuite envoyer un message au *team_server* pour lui indiquer que le robot en question a fait un tir à partir de la position du ballon dans la direction de l'avancement du robot, le *team_server* envoi alors cette commande au logiciel de simulation de mouvement du ballon (*soccerfield_sim*). Dans le cas du vrai robot, le système pneumatique est utiliser pour botter le ballon Il faut donc élaborer un modèle qui nous donne la bonne intensité de coup de tir pour que le ballon arrive à sa cible avec une erreur de 0.2 mètre ayant une vitesse assez faible pour que le receveur puisse aisément attraper le ballon.

Tout comme dans le cas d'interception du ballon, deux partons ont été crée pour cet entraînement, celui de « *ShotBallLearning* » et celui de « *ShotBallLearned* ». Le patron « *ShotBallLearning*» correspond au scénario de test décrit précédemment, tandis que le patron « *ShotBallLearned* » utilise le modèle crée par le patron de « *ShotBallLearning*» afin de prédire la bonne vitesse que le robot doit prendre pendant un match.

Les entrée et sortie de ce modèle d'apprentissage sont définies de façon suivantes :

Entrée	Sortie
$X = [Distance_{ballon\_cible}, Angle_{ballon\_cible}]$	Y = [Intensité du coup]

Tout comme dans le cas d'interception du ballon, un ajustement des paramètres du modèle d'apprentissage (C, epsilon, paramètres du *Kernel*), a été effectué.

3.6.3. Résultat pour l'apprentissage par la méthode SVM et MLP

Les méthodes d'apprentissages utilisées ont été celles de l'apprentissage par SVM (par régression) et aussi celle de l'apprentissage par MLP (par régression). Pendant chaque séance de test, 2000 tirs ont été effectués.

- Pour l'apprentissage par SVM, le meilleur taux de réussite trouvé n'était pas plus que 39%. Une discrétisation de 10 cm a été utilisé pour la distance du ballon, 5 degré pour l'angle entre le ballon et le cible. Les paramètres utilisés (après ajustement) pour le modèle SVM ont été : un σ de 10, un facteur de compromis (C) de 100, l'epsilon (ε) de 0.7 ainsi que un nombre de données d'entraînement après discrétisation de 700.

- Pour l'apprentissage par MLP, avec le même nombre de donnée d'entraînement soit 700, plusieurs nombres d'unité cachés ont été choisis. Les taux de réussites suivantes ont été obtenus :
    - Un taux de 28% avec un nombre caché de 3
    - Un taux de 33% avec un nombre caché de 4
    - Un taux de 45% avec un nombre caché de 5
    - Un taux de 48% avec un nombre caché de 7
    - Un taux de 40% avec un nombre caché de 10
    - Un taux de 65% avec un nombre caché de 20

On remarque que pour ce comportement avec plus de 4 unités cachées, l'apprentissage par MLP donne des résultats supérieurs comparés à celui par SVM.

La figure suivante nous illustre l'intensité du tir appris par les méthodes SVM et MLP avec 20 couches en fonction de la distance et l'angle entre le robot et sa cible.

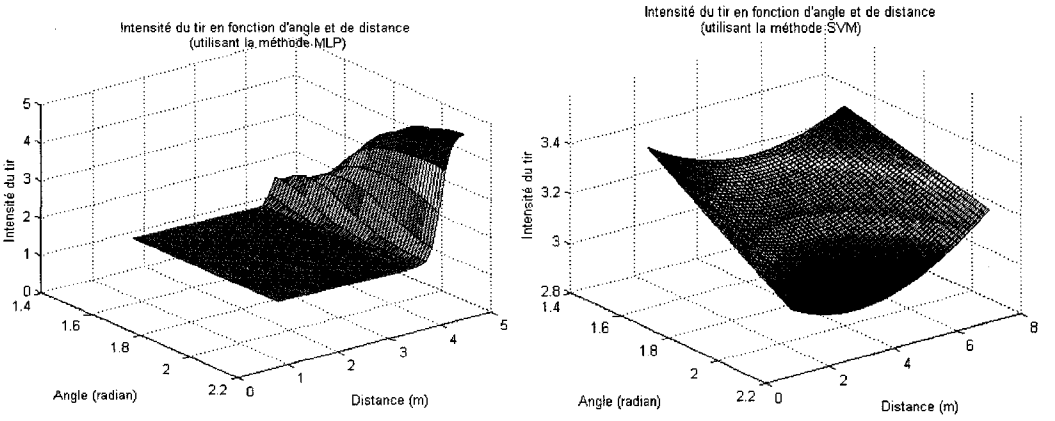


Figure 3.10 : Graphique de l'intensité du tir en fonction de l'angle et la distance entre le robot et sa cible utilisant la méthode d'apprentissage de SVM et MLP

On peut remarquer que la méthode d'apprentissage par MLP donne un graphique qui paraît être plus cohérant que celui de SVM. Pour la méthode de SVM, nous avons une surface donc la valeur d'intensité se limite entre 3 et 3.4 tandis que pour la méthode MLP l'intensité semble être constante pour des distance entre assez petite (vraisemblablement assez pour éliminer l'effet de la friction et assez modéré pour atteindre la cible à une vitesse acceptable) et augmente par la suite.

3.7. Apprentissage du comportement de la protection du but (ProtectGoal)

Un des joueurs le plus important dans une équipe est le gardien. Dans la machine décisionnelle hiérarchisée élaborée par Julien Beaudry, le comportement principal d'un gardien est celui de protection du but ou « *ProtectGoal* ».

3.7.1. Scénario d'entrainement

Le principe du comportement « *ProtectGoal* » est simple en fait il s'agit de mettre le gardien entre le but et le ballon. Tout comme dans le cas de l'interception du ballon, il faut d'abord faire un entraînement pour recueillir assez de données pour pouvoir bâtir un modèle. Le scénario choisi est semblable au cas précédant c'est-à-dire que l'on effectue des tirs vers le but que le gardien tentera de bloquer. Le paramètre à apprendre dans ce cas est celui de la vitesse que le gardien doit prendre afin de s'assurer d'attraper le ballon.

Dans ce cas, les deux comportements crées sont celui de « *ProtectGoalLearning* » et celui de « *ProtectGoalLearned* ». Le patron « *ProtectGoalLearning* » correspond au scénario de test décrit précédemment, tandis que le patron « *ProtectGoalLearned* » utilise le modèle crée par le patron de « *ProtectGoalLearning* » afin de prédire la bonne vitesse que le robot doit prendre pendant un match afin d'attraper le ballon.

Les entrée et sortie de ce modèle d'apprentissage sont définie de façon suivantes :

Entrée	Sortie
$X = [Distance_{ballon\_cible}, PositionInitial_{gardien}, Angle_{ballon\_cible}]$	Y = [Vitesse à prendre]

La figure suivante montre les commandes de tir envoyées au robot, et le robot qui par la suite demande au serveur la vitesse à prendre pour arrêter le ballon.

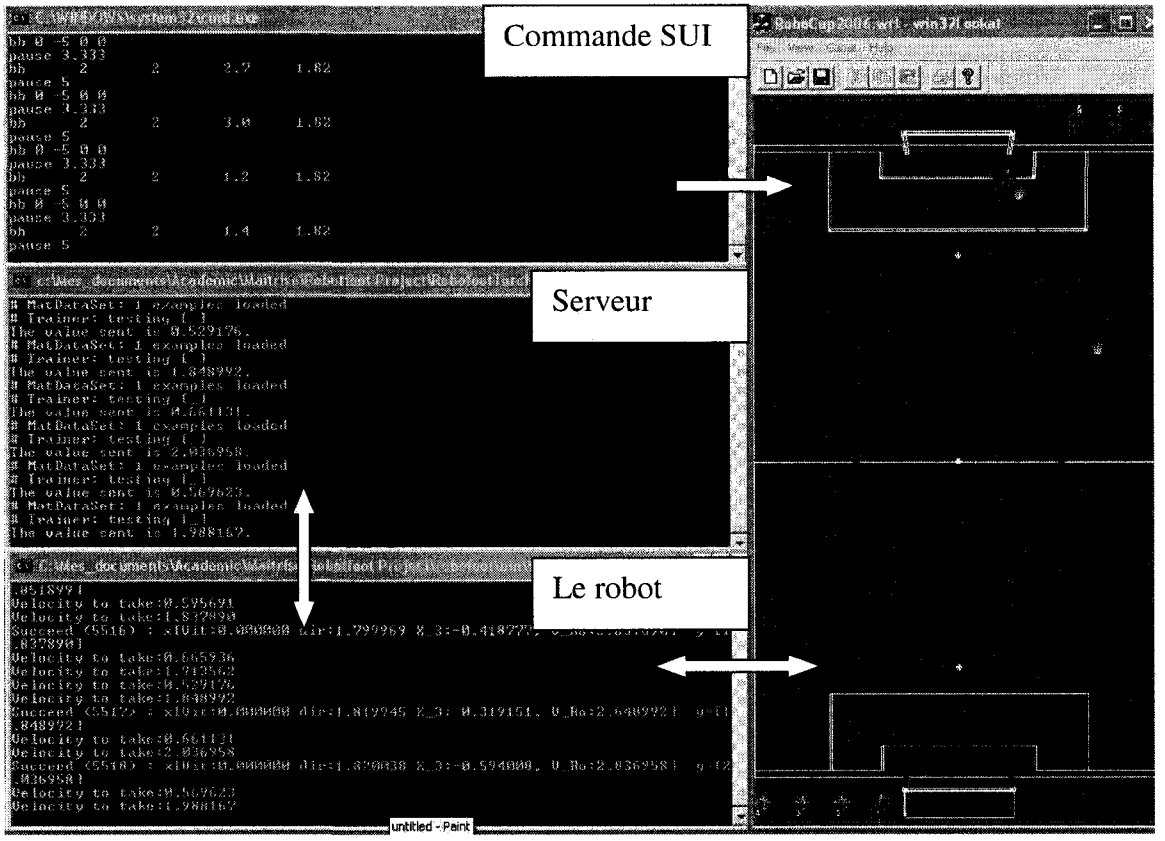


Figure 3.11: Principe du comportement de protection de but après l'apprentissage

3.7.2. Résultat pour l'apprentissage par la méthode SVM et MLP

Les méthodes d'apprentissages utilisées ont été celle de l'apprentissage par SVM (par régression) et aussi celle de l'apprentissage par MLP (par régression). Pour l'apprentissage un total de 10000 essais a été effectué pour des vitesses de ballon allant de 1 m/s à 3m/s pour tous les angles devant le robot. Le taux de réussite avant l'apprentissage a été de 45.90%.

- Pour l'apprentissage par SVM, le taux de réussite a augmenté à 89.86%. Une discrétisation de 10 cm a été utilisé pour la position du robot ainsi que pour la distance du ballon, 5 degré pour l'angle entre le ballon et le cible et 0.1 m/s pour la vitesse. Les paramètres utilisé pour le modèle SVM été : un σ de 10, un facteur de compromis (C) de 200, l'epsilon (ε) de 0.7 ainsi que un nombre de donnée d'entraînement après discrétisation de 1900.

- Pour l'apprentissage par MLP, avec le même nombre de donnée d'entrainement soit 1900, plusieurs nombres d'unité cachés ont été choisis. Avec un nombre de couche caché plus grand que 5 notre modèle devient trop grand pour pouvoir fournir des résultats assez rapidement sur ce système en temps réel. En résumé, après avoir fait 800 essais avec des différents nombres de couche cachés, les taux de réussites suivants ont été obtenus :

    - Un taux de 51.54% avec un nombre d'unité cachée de 5
    - Un taux de 76.29% avec un nombre d'unité cachée de 4
    - Un taux de 62.68% avec un nombre d'unité cachée de 3
    - Un taux de 62.50% avec un nombre d'unité cachée de 2

Ce qui nous permet de confirmer qu'avec 4 unités cachés on a le meilleur taux de succès pour cet apprentissage.

On conclut donc que l'apprentissage par SVM a donnée encore des résultats plus satisfaisants par rapport à l'apprentissage par MLP.

3.7.3. Résultats expérimentaux

Sur les robots, une séance de 300 tirs a été effectuée en essayant d'avoir une vitesse entre 1m/s à 3m/s. La séance d'entrainement ainsi que les résultats obtenus sont décrit dans la section 4.7.3. En résumé, un taux de réussite de 71% a été achevé utilisant la méthode SVM.

3.8. Apprendre un comportement collectif

Tout comme des vrais joueurs de soccer, après avoir appris à un manipuler le ballon de manière appropriée, nos robots doivent maintenant apprendre un comportement qui nécessite une coopération entre deux ou plusieurs joueurs. La tâche la plus élémentaire qui nécessite la coopération des robots s'agit de reçevoir une passe.

3.8.1. Définition d'une passe

D'abord, il faut préciser ce qui constitue une passe appropriée dans le cadre de ce travail. On peut définir une passe appropriée comme étant un tir réussi conformément à la définition donnée dans la section 3.6.1 suivi d'une interception réussi conformément à la définition donnée à la section 3.4.2. Aussi, si le receveur est devant le ballon, la passe doit s'effectuer de sorte que le ballon se trouve devant le receveur après le tir et non pas en arrière de lui, ce qui limite les choix de l'angle de celui qui fait la passe. Par contre, si le receveur se trouve derrière le ballon, la passe peut se faire à n'importe quel angle autant que l'on est assuré que le ballon se trouvera dans la zone devant le robot.

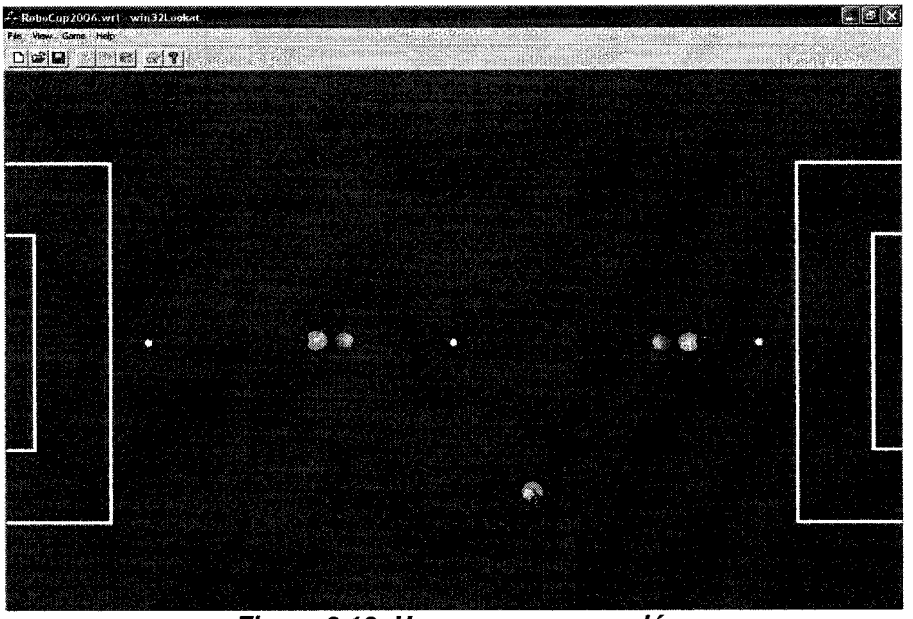


Figure 3.12: Une passe appropriée

3.8.2. Scénario d'entraînement en simulation

Tout comme dans le cas de l'interception du ballon, il faut d'abord faire un entraînement pour recueillir assez de données pour pouvoir bâtir un modèle. Dans ce cas, une première tentative a été faite sans joueurs adverses et seulement en combinant les deux comportements ensemble, et ensuite une deuxième tentative a été fait avec la présence de trois joueurs adverses.

La passe est une réussite si le receveur réussi à faire avancer le ballon vers le but et un échec si l'adversaire réussi à intercepter le ballon ou encore si le ballon sort de la zone visée sans interception.

Afin de faciliter le développement de ce scénario un nouveau mode a été ajouté à la machine décisionnelle hiérarchique conçue par Julien Beaudry. On peut voir sur la figure 4.11 la hiérarchie associée à ce nouveau mode, ou le premier étage correspond à l'étage du mode décrit dans le mémoire de maîtrise de Julien Beaudy(2005).

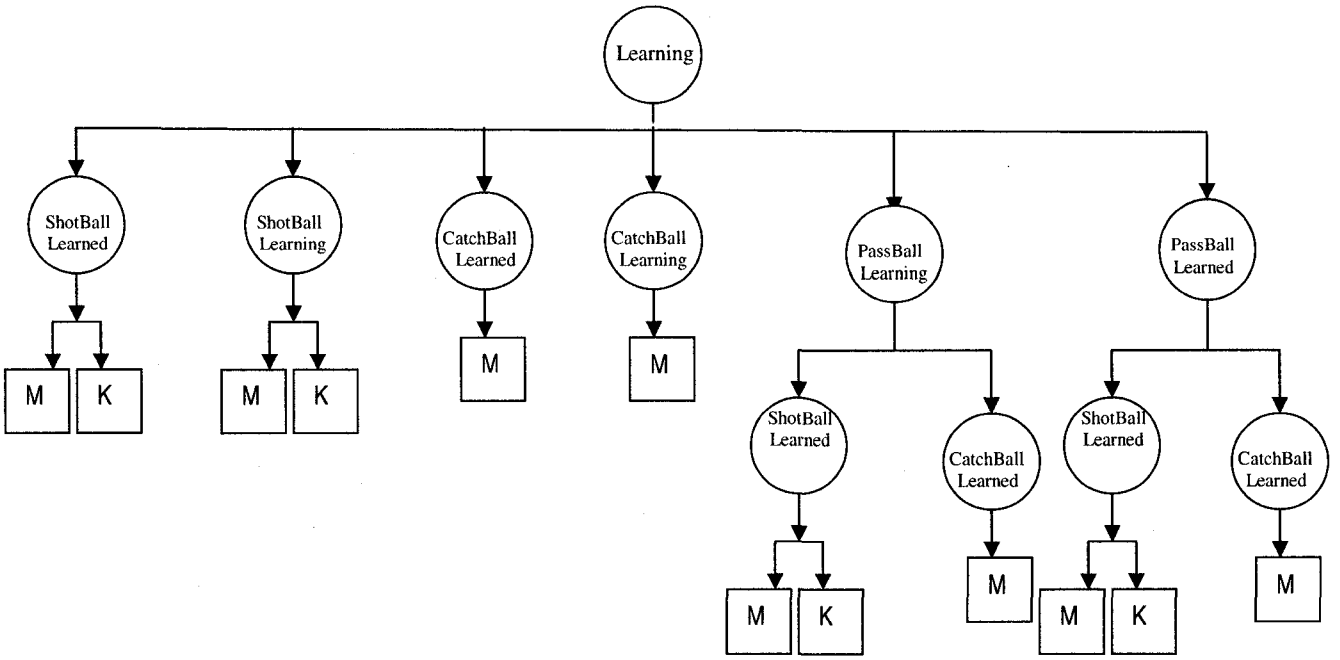


Figure 3.13 : Schéma de l'hiérarchie associée au mode « Learning » (M : Move, K : Kick)

Le patron « *PassBallLearning* » correspond au scénario de test utilisé pour combiner les comportements « *ShotBallLearned* » et «*CatchBallLearned*» sans adversaire et le patron « *PassBallLearned* » correspond au scénario de test avec adversaire.

Les patrons « *PassBallLearned* » correspondent au patron qui utilise le modèle bâti par le patron « *PassBallLearning* ».

3.8.2.1. Sans adversaire

Pour ce cas, seulement une combinaison des deux comportements « *ShotBallLearned* » et « CatchBallLearned» a été fait. Lors de cette séance d'entraînement, un robot est chargé d'aller porter le ballon à une position vers le centre du terrain et ensuite de faire un tir vers le deuxième robot. Le deuxième robot a

la tâche de faire une interception de cette passe et de retourner à sa position initiale afin d'attendre le prochain tir. Contrairement aux séances précédentes, cette séance d'apprentissage est entièrement automatique et ne requiert aucune intervention d'une tierce partie (*SUI*).

Dans ce cas, c'est le robot qui interceptant le ballon qui trouve le taux de réussite. Un taux de réussite de 18% a été obtenu pour une simulation de durée de 72 heures consécutive. Il est important à noter qu'ayant un taux de succès de 49% pour l'interception du ballon et 65% pour les tirs, on ne pourrait s'attendre qu'à un taux de réussite maximal de 32% (0.49 * 0.65).

3.8.2.2. Avec adversaire

Le scénario choisi en simulation pour faire les tests (l'entraînement du notre modèle) consiste à faire une tentative de passe par un joueur (passeur) qui a à sa droite deux joueurs amis (un plus proche du but adverse que lui et un plus loin du but adverse que lui) qui sont placés au hasard dans les zones 1 et 2 de la figure 3.14. Il y a aussi 3 joueurs adverses qui sont placés au hasard dans les zones 1, 2 et 3 défini dans la figure suivante et ils font des mouvements sur une ligne verticale et la passe se fait au hasard en terme du temps de l'envoi du ballon vers le receveur.

Le passeur va alors choisir au hasard à faire une passe à un des receveurs.

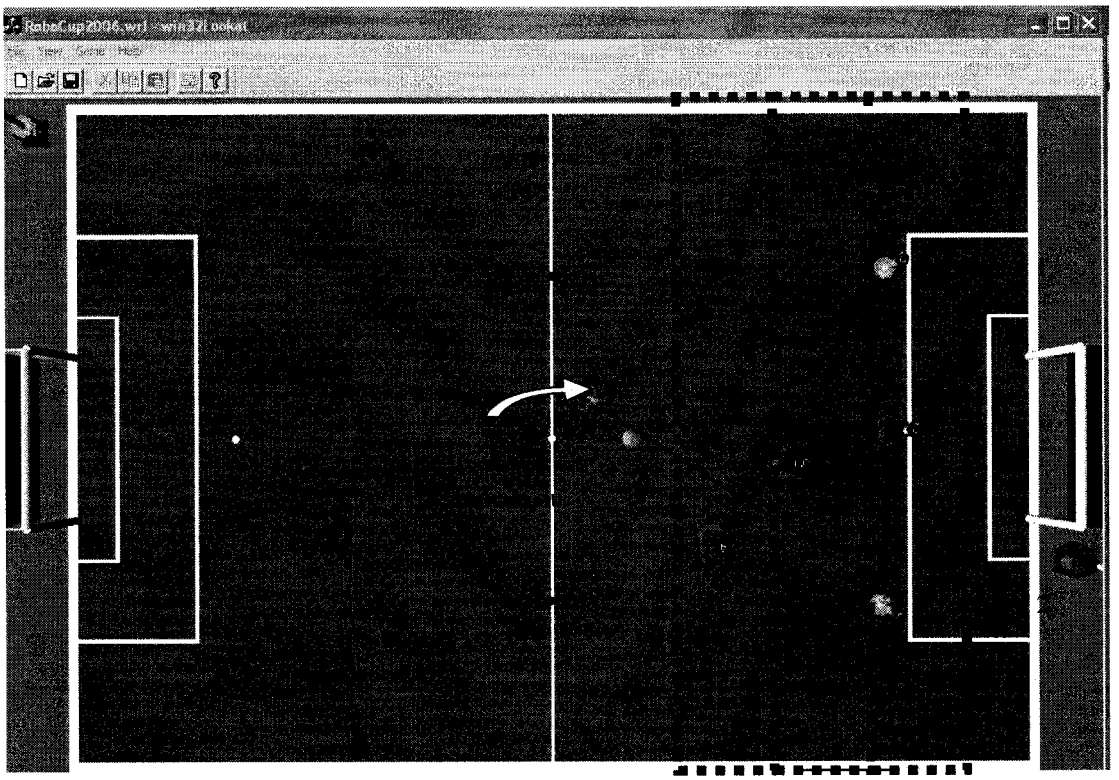


Figure 3.14 : Scénario d'entraînement pour apprendre à faire une passe

Les joueurs amis vont alors essayer de faire une interception du ballon en se servant du modèle bâti lors de l'apprentissage de ce comportement

La méthode d'apprentissage par prédiction SMV par classification a été utilisée pour choisir entre les deux receveurs. Le robot en possession du ballon recueillit toutes les informations nécessaires et les fait parvenir au serveur *RobofootTroch* (module d'apprentissage) qui va alors prédire vers quel receveur la passe doit être effectuée pour s'assurer d'une réussite. Cette valeur prédite sera ensuite renvoyée au robot pour qu'il puisse agir en conséquence.

Les entrée et sortie de notre modèle d'apprentissage sont définies de la façon suivante :

Entrée	Sortie
X = [Zone $_{receveur1}$,	Y = [0, //Aucune passe
Zone $_{receveur2}$,	1, //Passe au receveur1
Zone $_{defenseur1}$,	2 //Passe au receveur2
Zone $_{defenseur2}$,	]
Zone $_{defenseur3}$,]	

Où la zone de chaque joueur est une fonction de sa position. En fait, afin de bien discrétiser notre modèle d'état, le terrain qui mesure 8 m * 12 m, est divisé en morceau de 25cm^2, créant ainsi 1536 morceaux de terrains qui ont chacun un numéro qui leur a été assigné.

Chacun des receveurs sont positionnés dans une moitié de terrain, leurs positions peuvent changer de façon aléatoire dans une moitié de terrain, la position du ballon change aussi utilisant un fichier de script qui utilise le *SUI* pour communiquer avec *soccerfield_server.*

Tout comme dans le cas d'interception du ballon, il faut faire un ajustement des paramètres du modèle d'apprentissage (C, epsilon, paramètres du *Kernel*).

Il est à noter qu'un seul ordinateur (CPU double 2GHz avec 1GB RAM) a été jugé comme ne pas étant assez puissant pour faire cet apprentissage en temps réel, vu la quantité de calculs demandés en même temps du serveur à partir des trois clients afin de supporter cette séance.

3.8.3. Scénario d'entraînement expérimentaux

Le scénario d'entraînement de passe sur les vrais robots a été fait sans la présence des joueurs adverses et le comportement de « *ProtecGoalLearned* » a été utilisé pour intercepter le ballon à cause de son taux de succès plus élevé. Lors de cette séance d'entraînement, tout comme en simulation un robot est chargé d'aller porter le ballon à une position vers le centre du terrain et ensuite de faire un tir vers le deuxième robot qui va alors l'intercepter. Deux robots de type différentiel ont été utilisés pour faire ce test. La partie de code pour ramener le ballon vers un point donnée avec la précision voulu a été modifiée par rapport à celui utilisé en simulation pour tenir compte des imprécisions du robot.

Dans le cas où le ballon reste immobile à une position donnée (dans un coin du terrain), le robot passeur produira à chaque 20 secondes un signal sonore pour laisser savoir que le test a été bloqué et qu'il y a besoin d'une intervention humaine, mais malheureusement aucun des robots actuels n'est équipé d'une carte de son (ce qui ne sera pas le cas pour le prochain robot) et donc un message texte dans la console a été ajouté pour aviser l'usager pour le moment. La séance d'entraînement a été exécutée de façon semi-automatique (une l'intervention humaine pour sortir le ballon des situations laborieuse a été nécessaire dans presque 40% des cas). Un taux d'interception de 71% a été observé sur une série de 300 tirs utilisant le modèle crée par l'apprentissage par SVM. Ce qui est 15% moins que les résultats obtenu en simulation.

Il faut aussi noter que dû à l'imprécision associé au système de vision une partie des tirs était manqué, c'est-à-dire que le robot tirait trop tôt (donc aucun contact avec le ballon) ou n'arrivait pas à tirer (la position du ballon était vue plus loin que sa position réel). Les tirs manqués n'ont pas été pris en compte dans les résultats mentionnés.

3.9. Apprentissage au niveau d'équipe

Après avoir appris un comportement collectif de base, on peut alors faire apprendre à l'équipe des robots joueurs de soccer les patrons de jeu et les différentes formations qui sont les plus adaptés en fonction d'une situation donnée. Étant donnée l'envergure de cet apprentissage ainsi que la non disponibilité de deux équipes complètes, dans le cadre des travaux effectué, ce niveau d'apprentissage a seulement été étudié en simulation.

3.9.1. Patrons

L'architecture de décision choisie par J. Beaudry consiste en fait à une **Machine décisionnelle hiérarchique.** En résumé cette machine permet de choisir un patron prédéfini qui prend des décisions hiérarchisées. Ces décisions permettent de choisir des comportements qui à leur tour choisissent des actions en mettant à jour leurs paramètres.

La condition qui permettait de choisir le rôle d'un robot est très simple; si le ballon est à une distance plus grande que 1.5 mètre de nos robots ou s'il est dans notre zone on change de mode pour être dans le mode défensif, sinon si le robot est dans la zone adverse et que le ballon est proche de nos robots on bascule en mode offensif. La figure suivante montre les zones offensives et défensives.

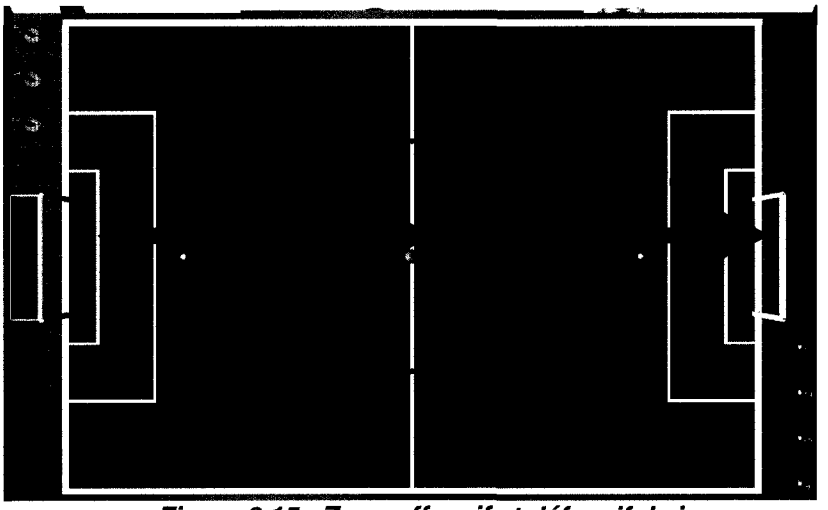


Figure 3.15 : Zone offensif et défensif de jeu

L'apprentissage s'effectue lors du changement de mode entre le mode offensif et défensif. C'est-à-dire que, à chaque fois que le mode de jeu change on choisit un patron offensif ou défensif différent. Trois patrons offensifs et trois patrons défensifs ont été choisis pour être comparés. La section qui suit expliquera brièvement le principe de fonctionnement de chacun de ces patrons.

- Patron « défensif avec les zones prédéfinies 1 »: *(DefensifRC)* ce patron conçu originalement par Julien Beaudry, consiste à désigner une zone à défendre définie pour chaque robot. Chacun des robots est assigné un des rôles présents pour ce patron soit « *DefenseZone1* », « *DefenseZone2* », « *DefenseZone3* » ou « *DefenseZone4* ». La figure suivante présent de façon visuelle ce patron.

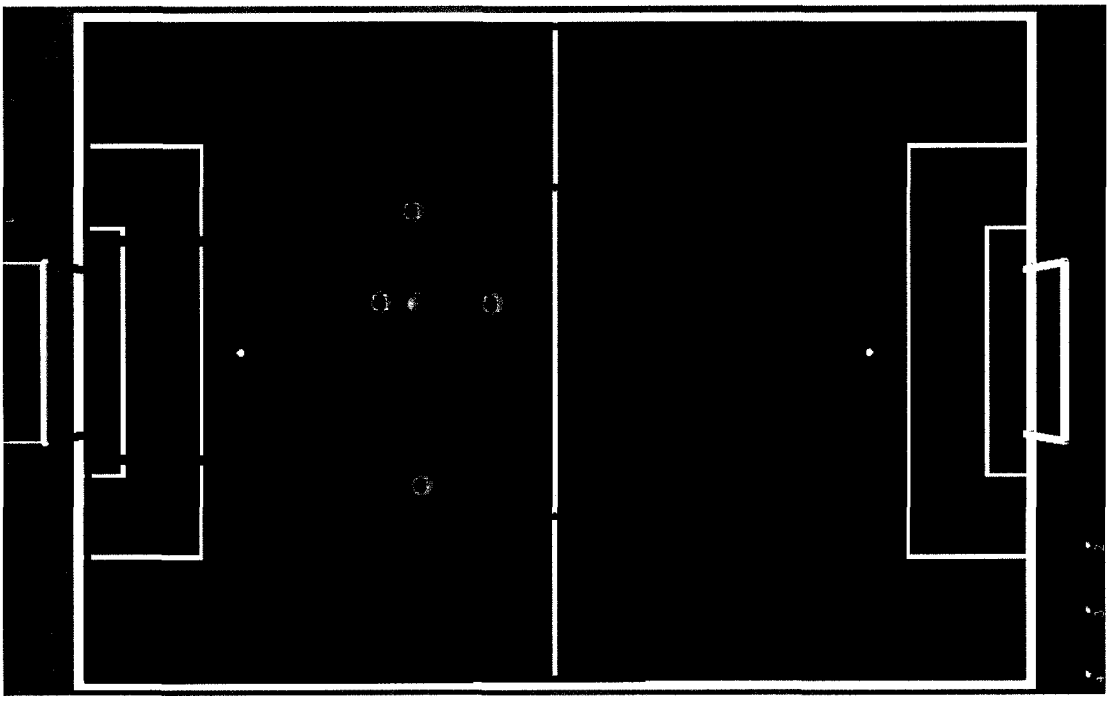


Figure 3.16 : Premier patron défensif dans les zones prédéfinies

- Patron « Défensif avec support » (*DefensifAR*) : ce patron consiste à donner à trois robots des zones prédéfinies à défendre dans le terrain, mais le quatrième robot a l'occasion d'aller dans la zone de ses coéquipier. Alors, si aucun des trois robots ayant des zones préétablies ne possèdent le ballon, le robot supporteur essaiera d'attraper le ballon sinon, il se placera derrière son coéquipier qui possède le ballon pour pouvoir venir à son aide dans le cas où celui-ci perd le ballon.

- Patron « défensif avec les zones prédéfinies 2 » (*DeffensifRC2) :* ce patron est semblable au premier patron, mais les robots ont des zones différents. La figure 3.17 permet de voir les zones utilisées pour ce patron.

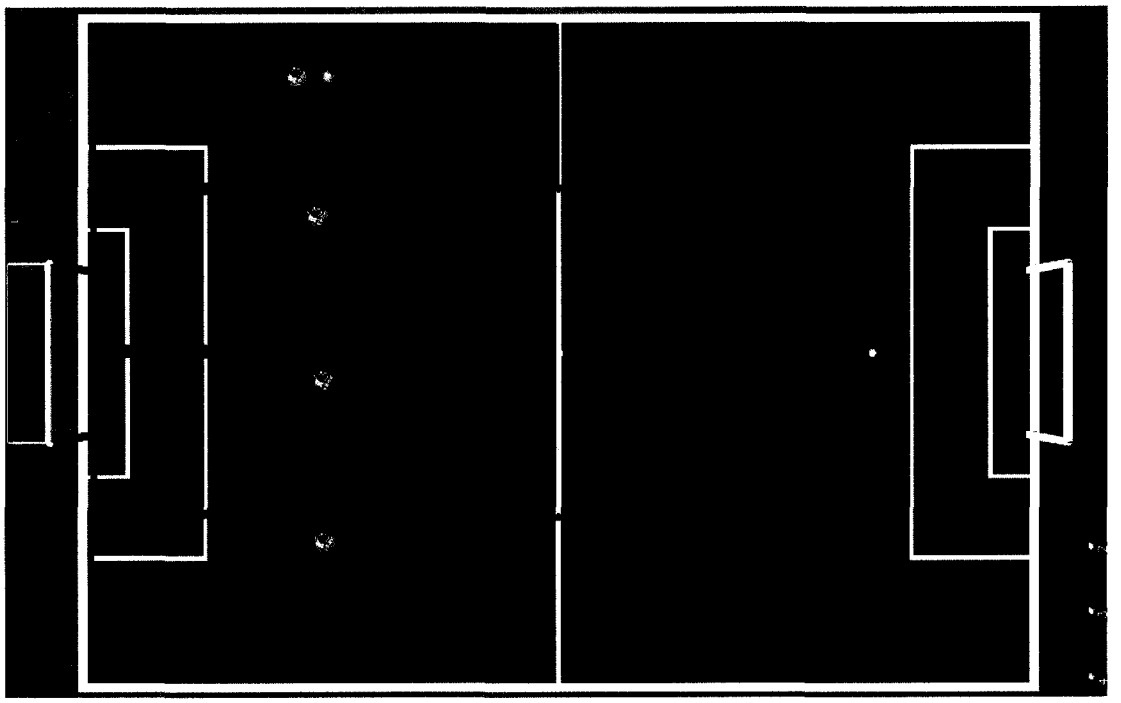


Figure 3.17 : Deuxième patron défensif dans les zones prédéfinies

- Patron « Offensif Leader-suiveur » (*OffenseRC*): ce patron originalement conçu par Julien Beadry, consiste à un leader et trois suiveur. En fait, le robot qui est le plus proche du ballon devient le leader, en essayant d'amener le ballon vers le but d'ennemi, alors les trois autres robots forment une formation de triangle à une distance de 2m derrière celui-ci. La figure 3.18 permet de visualiser ce patron.

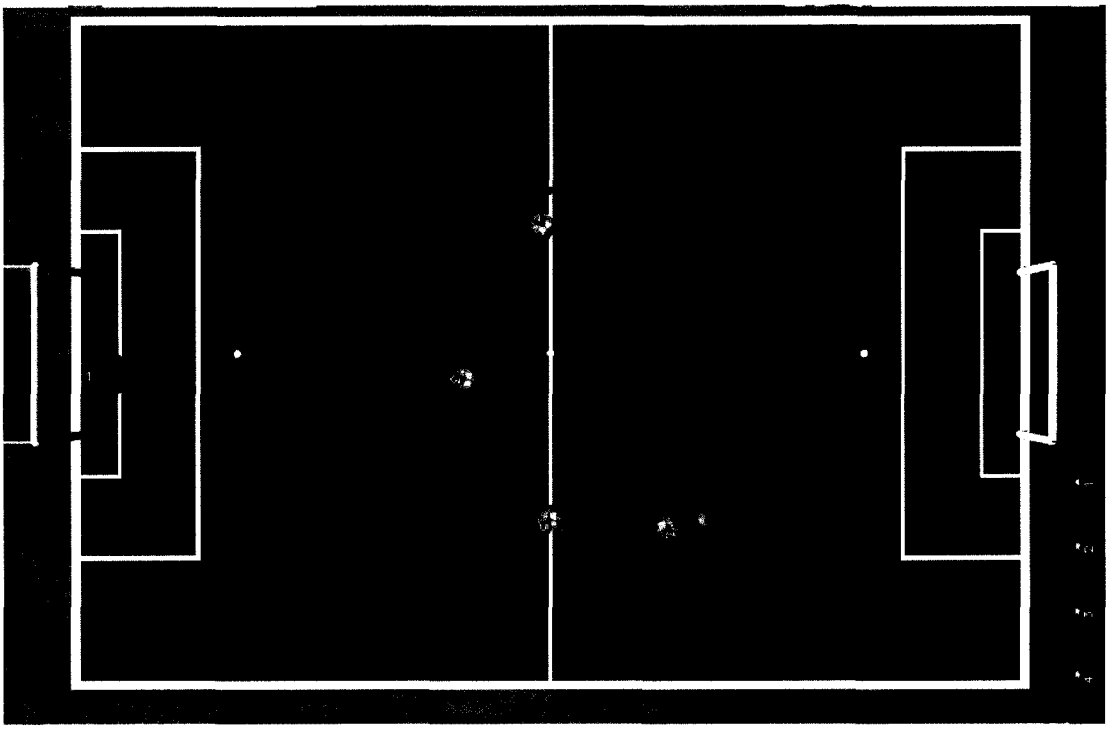


Figure 3.18 : Patron Offensif Leader-suiveur

- Patron « Offensif avec formation », (*OffenseAR*) dans ce cas 3 robots forment une ligne perpendiculaire dans la direction du ballon et le quatrième robot reste en position défensive à 3 mètre en arrière de la formation créé par les autres robots, une fois le ballon attrapé par un des robots, ils essayeront d'entourer le ballon avec une formation en triangle et avancer vers le but.

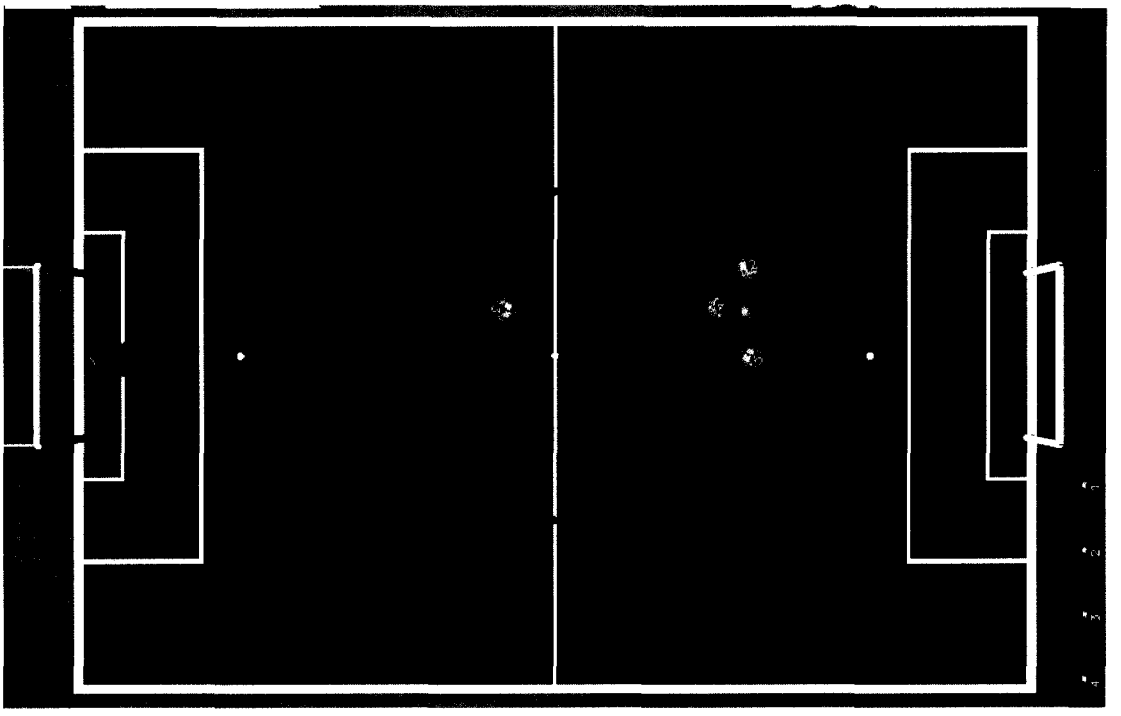


Figure 3.19 : Patron avec une formation triangle

- Patron « Offensif avec une formation agressive », (*OffenseAG*) dans ce cas, on a encore un robot qui le leader mais il est appuyé par un robot qui reste à une très courte distance de lui, et les 2 robots restants les suivent à une distance plus grande tel que démontré dans la figure suivante.

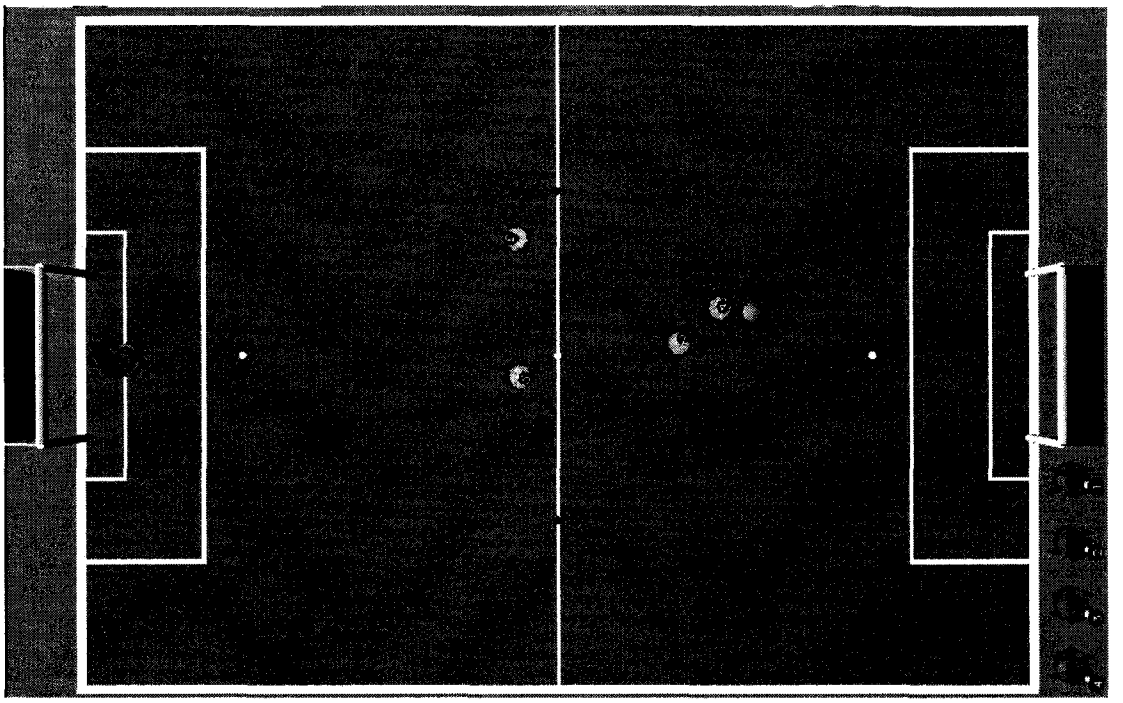


Figure 3.20 : Patron offensif agressif

3.9.2. Architecture d'apprentissage

Le scénario d'entraînement utilisé pour cette partie de ce travail consiste à tenter chacun des stratégies pendant soixante minutes. À la fin de chaque essai, les statiques suivantes sont recueillies et enregistré dans un fichier,

Nom du variable	Description
In_Opp_Field	Le temps passé dans la zone adverse
In_Own_Field	Le temps passé dans la zone ami
In_Opp_Field_Percentage	Pourcentage de temps passé dans la zone adverse
In_Own_Field_Percentage	Pourcentage de temps passé dans la zone ami
team1_gotBall	Le temps de possession du ballon pour l'équipe 1
team0_gotBall	Le temps de possession du ballon pour l'équipe 0
team1_gotBall_Percentage	Pourcentage de possession du ballon pour l'équipe 0
team0_gotBall_Percentage	Pourcentage de possession du ballon pour l'équipe 0
goalteam0	Le nombre de but pour l'équipe 0
goalteam1	Le nombre de but pour l'équipe 1

Tableau 3.4 - Statiques recueillies par chaque robot pendant un match

Après la période d'essai les résultats sont traités utilisant la formule suivante :

$$5\ (But_{pour} - But_{contre}) + (\%Possession_{pour} - \%Possession_{contre}) + (\%T_{\text{en zone adverse}} - \%T_{\text{en zone amie}})$$

Tel que l'on remarque la différence de but a été attribuée une grande importance par rapport au temps de possession et le temps passé dans le terrain adverse vue que ceci est le but ultime lors d'un match. C'est le logiciel *team_server* qui se charge de recueillir toutes les statistiques grâce à son accès global aux informations pertinentes de tous les robots. Ensuite, les statistiques sont enregistrées soit après une durée limitée de temps, ou au moment même que l'on change de patron.

Deux fonctions d'écriture de statistiques ont été développées, la première permet d'enregistrer des statistiques pendant un match contre un adversaire dont on ne connait pas les patrons et on utilise alors une contrainte de temps pour enregistrer les données. La deuxième fonction qui a été utilisée pour faire l'apprentissage entre les patrons interne et donc à chaque fois que l'on change de patron, on utilise le vecteur de décision d'un des robots pour savoir qu'un changement de patron a été fait pour les robots et il enregistre les statistiques correspondants à l'ancien patron.

Pour cet essai, deux équipes constituées une de robots omnidirectionnels et l'autre de robots différentiels jouent l'une contre l'autre en simulation. L'équipe omnidirectionnelle utilise l'apprentissage et l'équipe ayant des robots différentiels joue sans utiliser l'apprentissage.

Les résultats obtenus pour un essai est combiné avec les résultats obtenus pendants les essais précédents utilisant le même patron afin de permettre de prendre une décision sur le patron le plus approprié lors de cet essai.

Lors d'un match deux modes de sélections sont utilisés, un mode statique et un mode de sélection dynamique. Le mode statique consiste à prendre le meilleur patron

trouvé grâce aux critères mentionnées, donc si le patron choisi performe mal et qu'il devient deuxième dans le classement des patrons, on change de patron. Par contre dans le mode dynamique, on associe à chacun des patrons une probabilité de sélection qui sont égales au départ, mais qui changent pendant les séances d'entraînement en fonction d'apprentissage : les patrons performants ont une probabilité de sélection plus grande.

3.9.3. Résultats

Une douzaine de séances de durée de deux heures par patron ont été simulées pendant deux semaines. Tel que mentionné dans la section précédente, les statistiques ont été enregistrées dans un fichier à la fin de chaque séance de deux heures pendant laquelle un patron donné a jouée contre un patron fixe soit le patron « *OffenceRC* ». La moyenne de ces résultats est présentée dans le tableau suivant.

Patron	Moyenne des statistiques
DefenseRC	19.56
DefenseAR	23.13
DefenseRC2	21.47
OffenseRC	41.44
OffenseAR	49.05
OffenseAG	105.91

Tableau 3.5 – Le statistique des différents patrons joués contre un patron fixe

On peut donc conclure que les patrons défensifs ont des performances assez similaires, et que le patron « *DefenseAR* » est légèrement supérieur par rapport aux autres patrons défensifs. Par contre pour les patrons offensifs, on constate que le Patron « *OffenseAG* » est nettement supérieur par rapport aux autres patrons offensifs.

Également, le mode de sélection statique a été comparé au mode sélection dynamique. Deux équipes composées de cinq robots omnidirectionnels se sont affrontées en simulation pendant une période de trois jours consécutifs. Il est à noter que l'équipe utilisant le mode de sélection statique a choisi le patron « *OffenceAG* » pendant toute la durée du match.

L'équipe utilisant le mode de sélection statique a gagné par un pointage de 120 contre 104. Aussi, il est intéressant de mentionner que la probabilité de choisir le patron « *OffenceAG* » dans le mode de sélection dynamique a monté de 16.6% à 41% à la fin

de ces entrainements. Ce qui confirme bien que ce patron est de loin le patron le plus dominant. La figure 3.21 illustre une partie joué entre deux équipe dont une choisi le patron utilisant le mode dynamique et l'autre utilisant le mode statique.

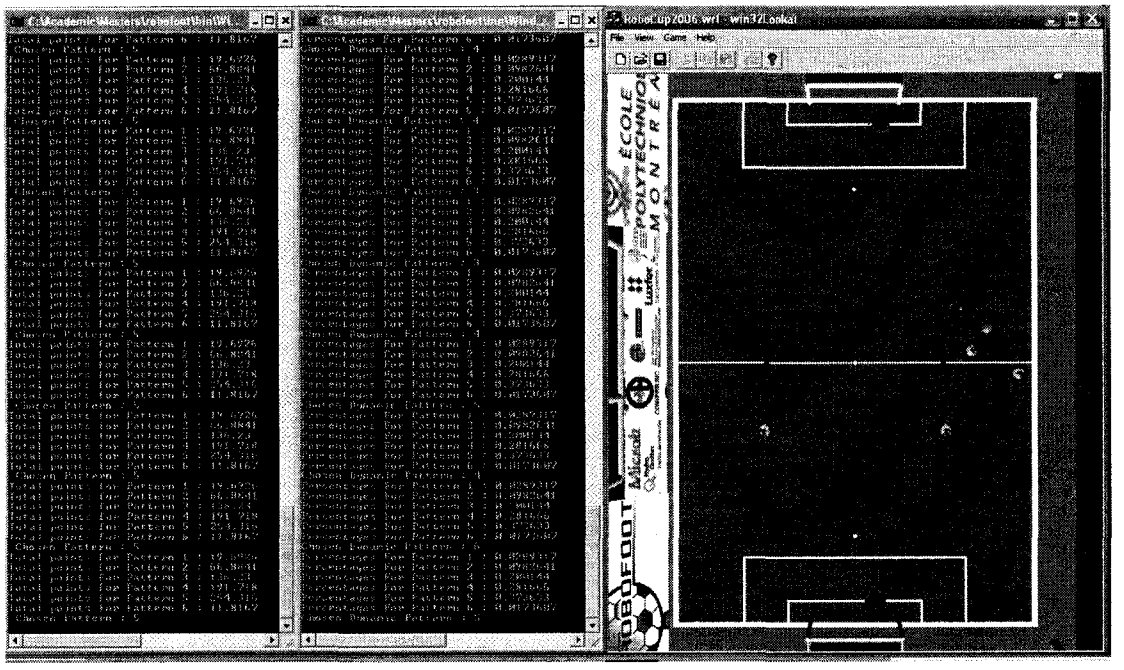


Figure 3.21 : Le mode de sélection dynamique contre le mode statique

Afin de pouvoir analyser l'évolution de la probabilité de sélection d'un patron dans le mode dynamique les trois meilleurs patron ont été raffiné pour jouer contre le mode statique, le graphique suivante montre l'évolution de la probabilité de sélection entre les patron *OffenceAR*, *OffenceRC* et *OffenceAG*.

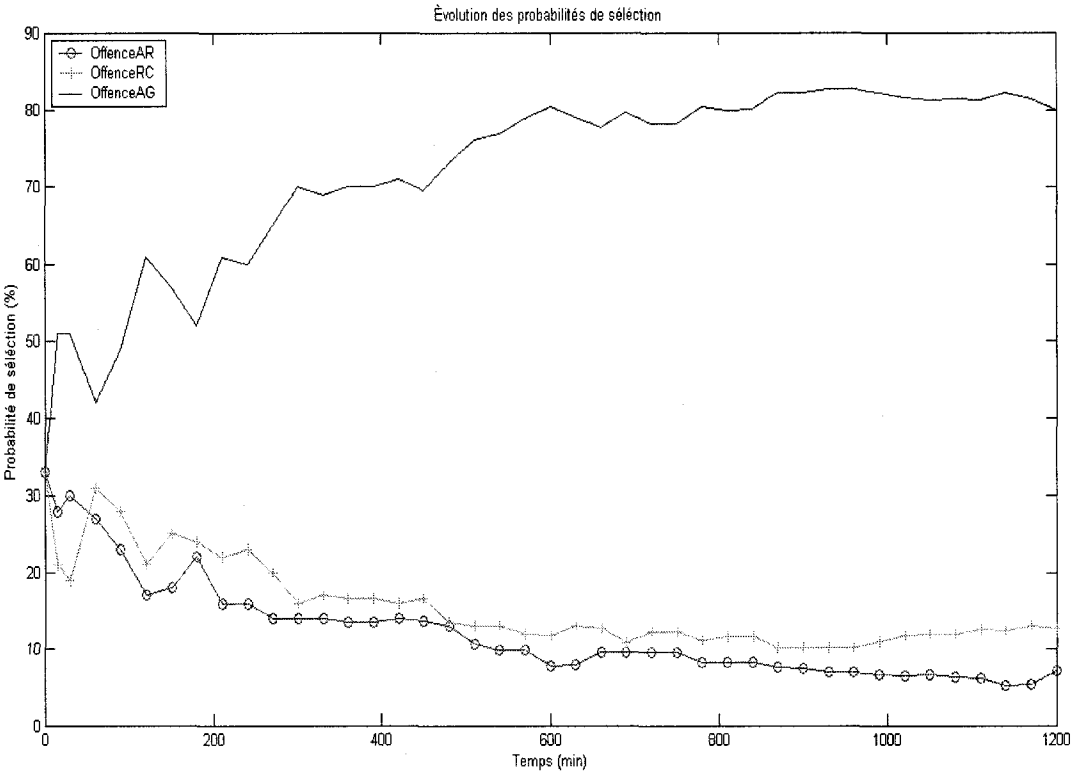


Figure 3.22 : L'évolution des probabilités de sélection parmi les patrons offensifs

On remarque une fois de plus que la probabilité de sélection du patron *OffenceAG* monte rapidement avec le temps pour atteindre un seuil d'environ 80%.

Il est intéressant de mentionner qu'en augmentant le nombre des patrons à sélectionner, la convergence de la probabilité de sélection d'un patron donnée vers une valeur quasi fixe devient plus lent. Par exemple, en comparant six patrons versus trois le temps de convergence pour avoir des probabilités de sélection stables est quasiment dix fois plus élevé.

Chapitre 4
Conclusion et travaux futurs

4.1. Conclusion

Les travaux réalisés dans le cadre de ce projet de maitrise ont été effectués pendant deux années. Garder plusieurs robots joueurs de soccer fonctionnels pendant une longue période de temps est assez complexe, et faire apprendre ces robots joueurs de soccer n'est pas plus simple. Alors, il a fallu travailler avec acharnement pour non seulement garder les robots dans un état fonctionnel mais aussi afin de développer ce système d'apprentissage générique, même si le système a été largement bien analysé et testé en simulation.

Ce projet de maîtrise a permis de développer un système d'apprentissage machine qui pourrai être utilisé dans d'autres domaines qui nécessitent une équipe de véhicules autonomes.

L'approche utilisée correspond à une architecture générique multi niveaux qui a permis d'aborder des problèmes (comportements) plus simple d'abord et ensuite les regrouper pour faire apprendre un comportement de plus haut niveau. Ceci est dû au fait que l'architecture générique a permis de bâtir un système entrée-sortie. L'architecture utilisée a aussi permis d'étudier différents algorithmes d'apprentissage qui ont été utilisés pour chaque niveau.

Ce projet de maitrise a permis d'étudier trois différentes méthodes d'apprentissage; celle de la méthode d'apprentissage par machine à vecteur de support (Support Vector Machines), celle de la méthode d'apprentissage par réseau de neutrons (Multi Layer Perceptron), ainsi que celle d'un apprentissage basé sur mémoire. Les

méthodes SVM et MLP ont été comparés entre elles et la méthode de SVM a permis de produire des résultats supérieurs par rapport à MLP dans la plupart des cas. Pour l'apprentissage des patrons de jeu la méthode d'apprentissage basé sur mémoire a été utilisée dû à la complexité de l'apprentissage et l'impossibilité de faire ceci avec une méthode basée sur des modèles mathématiques (tel que SVM ou MLP).

Dans le cas des apprentissages par SVM et MLP, il faut aussi souligner le fait que pour des comportements plus complexes tel que l'apprentissage d'une passe en présence des joueurs adverses, nécessitant la combinaison de plusieurs comportements appris, le temps des calculs des résultats combiné avec le délai client-serveur ne sont plus négligeables. Ceci fait probablement exposer la nécessité d'avoir un système distribué dans lequel plus qu'un serveur sert à répondre aux requêtes des robots.

En résumé, pour des comportements avec plusieurs entrées (trois et plus) tel que l'interception du ballon et la protection du but la méthode SVM a mené à des résultats plus avantageux tandis que pour l'apprentissage des comportements plus simple tel que le tir du ballon c'est l'apprentissage MLP qui a donné des résultats plus satisfaisants.

4.2. Travaux à venir

L'apprentissage machine est une étude qui peut prendre des années à implémenter sur un système donnée. Il y a donc plusieurs aspects de l'apprentissage qui n'ont pas été abordé lors de ce travail de maîtrise, voici une liste d'éléments qui devraient faire l'objet de développements futurs.

4.2.1. Implémentation de l'apprentissage pour l'ensemble des comportements de HDM

Le travail réalisé dans ce projet de maîtrise peut être considéré intéressant dans le sens que pour tous les comportements actuels et futurs utilisés pour les robots joueurs de soccer, les paramètres les plus importants ou ceux qui sont incertains peuvent être déterminés par apprentissage. L'annexe I illustre la hiérarchie des comportements/patrons utilisée actuellement par les robots joueurs de soccer. Un apprentissage peut être réalisé au niveau de tous les éléments de cette machine décisionnelle hiérarchisé (HDM).

4.2.2. Amélioration du matériel (CPU et caméra)

L'apprentissage fait au niveau des vrais robots a été limité par deux facteurs :

- l'imprécision des caméras utilisées par les robots : en fait le nombre de l'image par sec des caméras utilisées était à peine assez pour capter 3 à 4 images lors d'un tir de ballon.
- la puissance des ordinateurs embarqués sur les robots : l'apprentissage de certains comportements prend beaucoup d'espace disque pour sauver les essais et aussi un CPU assez puissant pour traiter le modèle crée par les méthodes SVM ou MLP. Une discrétisation a été fait afin de remédier à ce

prblème, mais ceci a comme conséquence de diminuer la précision de la réponse trouvée par le système d'apprentissage.

Alors la prochaine étape consiste à faire une mise à jour des caméras et des ordinateurs embarqués utilisés par les robots afin de permettre plus de flexibilité et de précision.

4.2.3. Implémentation d'un mécanisme pour diviser le calcul de l'apprentissage en temps réel

L'apprentissage par les méthodes de SVM et MLP demandent la création des modèles assez complexe qui peuvent prendre un certain temps avant de fournir le résultat. Pour un système en temps réel tel que celui des robots joueurs de soccer, il est très important d'avoir une réponse dans le plus court délai possible. Alors, il serait fort intéressant de développer un système de gestion pour distribuer les calculs demandés par les clients entre plusieurs serveurs.

Bibliographie

- Ronald C.Arkin, *Formation Behaviors, behavior-Base robotics*, The MIT Press, 1998

- Atekson, Moore et Schaal 1997, « Locally Weighted Learning for Control», Artificial Intelligence Review, February, Vol 11 No 1-5, pp 11-73

- J. Beaudry, Machine décisionnelle pour système multi-robots coopératifs, École Polytechnique de Montréal, 2005

- B. E. Boser, I. M. Guyon, and V. N. Vapnik. « A training algorithm for optimal margin classifiers», In Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, ACM Press, Pittsburgh, PA, 1992, pages 144–152

- L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK, 1998.

- Brooks, «*Intelligence Without Reason*», A.I. Memo No 1293, MIT AI Laboratory, 1991

- A. Cauchy. «Méthode générale pour la résolution des systèmes d'équations simultanées». In Compte Rendu Hebdomadaire des Séances de l'Académie des Sciences, volume 25, pages 536–538, Paris, France, 1847.

- R.-Commisso M.-A., Béliveau M. «*Conception et contrôle d'un robot omnidirectionnel* » PFE report, Département de génie électrique, École Polytechnique de Montréal. 2005

- Samuel Delepoulle, Philippe Preux, Jean-Claude Darcheville, «*L'apprentissage par renforcement comme résultat de la sélection*», *Université de Lille,* 2001

- Keith S. Decker and Victor R. Lesser. «*Designing a family of coordination algorithms*». In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), AAAI Press., June 1995. (pp. 195, 198)

- Denis F., Fillero R., Note du cours Intelligence artificielle à l'Université de Lille 3, http://www.grappa.univ-lille3.fr/polys/apprentissage/index.html, 2008

- Ronan Collobert, « *Large Scale Machine Learning* », thèse de doctorat, l'université de Paris VI , juin 2004

- Ronan Collobert, *Torch Tutorial*, IDIAP, http://www.torch.ch/matos/tutorial.pdf, October 2002

- N. Cristianini and J. Shawe-Taylor. An Introduction to Support Vector Machines. , Cambridge, University Press, 2000.

- G. W. Flake and S. Lawrence. « *Efficient SVM regression training with SMO.* » Machine Learning, 2002, 46(1–3):271–290.

- Y. Freund and R. E. Schapire. « *Large margin classification using the perceptron algorithm.* » Machine Learning, 1999, 37(3):277–296.

- K. Hornik, M. Stinchcombe, and H. White. « *Multilayer feedforward networks are universal approximators.* » Neural Networks, 1989, 2:359–366,

- J. J. Hopfield. « *Learning algorithms and probability distributions in feed-forward and feed-back networks.* » In Proceedings of the National Academy of Sciences, volume 84, pages 8429–8433, 1987.

- Y. LeCun. « *A learning scheme for asymmetric threshold networks.* » In Proceedings of Cognitiva 85, pages 599–604, Paris, France, 1985.

- T. K. Leen, T. G. Dietterich, and V. Tresp, editors, « *Advances in Neural Information* », Processing Systems, volume 13, pages 308–314. MIT Press, 2001.

- Tomás Lozano-Pérez, Lecture Notes, Artificial Intelligence, 2005, http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/ 6-034Spring-2005,

- S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K. R. Müller. « *Fisher discriminant analysis with kernels*», In Y. H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, Neural Networks for Signal Processing IX, pages 41–48. IEEE, 1999.

- McFarland D., The Oxford Companion to Animal Behavior, Oxford University Press
, 1981.

- N. J. Nilsson., Learning Machines. New York, McGraw-Hill, 1965.

- I. Rivals, L. Personnaz, G. Dreyfus, J.L. Ploix, Modélisation, «*Classification et commande par réseau de neurones : principes fondamentaux, méthodologie de conception et illustration industrielles*», École Supérieure de Physique et de Chimie Industrielles de la Ville de Paris Laboratoire d'Électronique.

- H. Robbins and S. Monro. «*A stochastic approximation method*», In Annals of Mathematical Statistics, volume 22, pages 400–407, 1951.

- Sims K. 1994, «*Evolving Virtual Creatures*», Proceedings of the ACM Siggraph 94, pp.15-22

- Simon H., «*Why Should Machines Learn*? », in Machine Learning : An Artificial Intelligence Approach, Col. 1, eds. R. Michalski, J. Carbonell, and T. Mitchell, Tioga Publishing, Palo Alto, CA, 1983, pp. 25-39

- R. Sutton and A. Barto. «*Reinforcement Learning: An Introduction*», MIT press, 2004.

- Sutton R., «*Reinforcement Learning and Artificial Intelligence*», University of Alberta, 2006, http://rlai.cs.ualberta.ca/RLAI/rlai.html

- Sutton R. and Santamaria J.C. , «*A Standard Interface for Reinforcement Learning Software in C++*»,Version: 1.1 2006, http://www.cs.ualberta.ca/~sutton/RLinterface/RLI-Cplusplus.html

- Stone, P., «*Layered Learning in Multiagent Systems*», The MIT Press, School of Computer Science, Carnegie Mellon University,Pittsburgh, 2000

- Szczap, Frédéric, «*Réseau de neurone*», cours de maitrise de deuxième année, 2005-2006, http://wwwobs.univ-bpclermont.fr/atmos/enseignement/cours-Master-2A/cours_RN_2006.pdf

- Milind Tambe. «*Towards fexible teamwork*», Journal of Artificial Intelligence Research, 7:81{124, 1997. (pp. 55, 56, 88, 89, 188, 192, 199, 205, 212)

- Torsten Soderstrom et Petre Stoica, «*System Identification*», Department of Technology, Uppsala University, Sweden, 1989

- Thondike E., Animal Intelligence, Hafner, Darien, CT, 1911

- V. Vapnik. The Nature of Statistical Learning Theory. Springer, second edition, 1995.

- V. N. Vapnik and A. Y. Chervonenkis. «*On the uniform convergence of relative frequenciesof events to their probabilities*», Theory of Probability and its Applications, 1971, 16(2): 264–280.

- E. ZENOU, «*Simulation de Systèmes Récurrents et Non Linéaires à l'Aide de Réseaux de Neurones*», Application au TA_IPAN, Juin 1998, p12-13

- Webster's Ninth New Collegiate Dictionary, Merriam-Webster, SpringField, MA., 1984

- «*Microb – Applications Manual* », Hydro Québec Institut de recherche, June 2005, http://www.robotique.ireq.ca/microb/en/index.html

ANNEXE I - *Machine Décisionnelle Hiérarchique* (tiré de J. Beaudry)

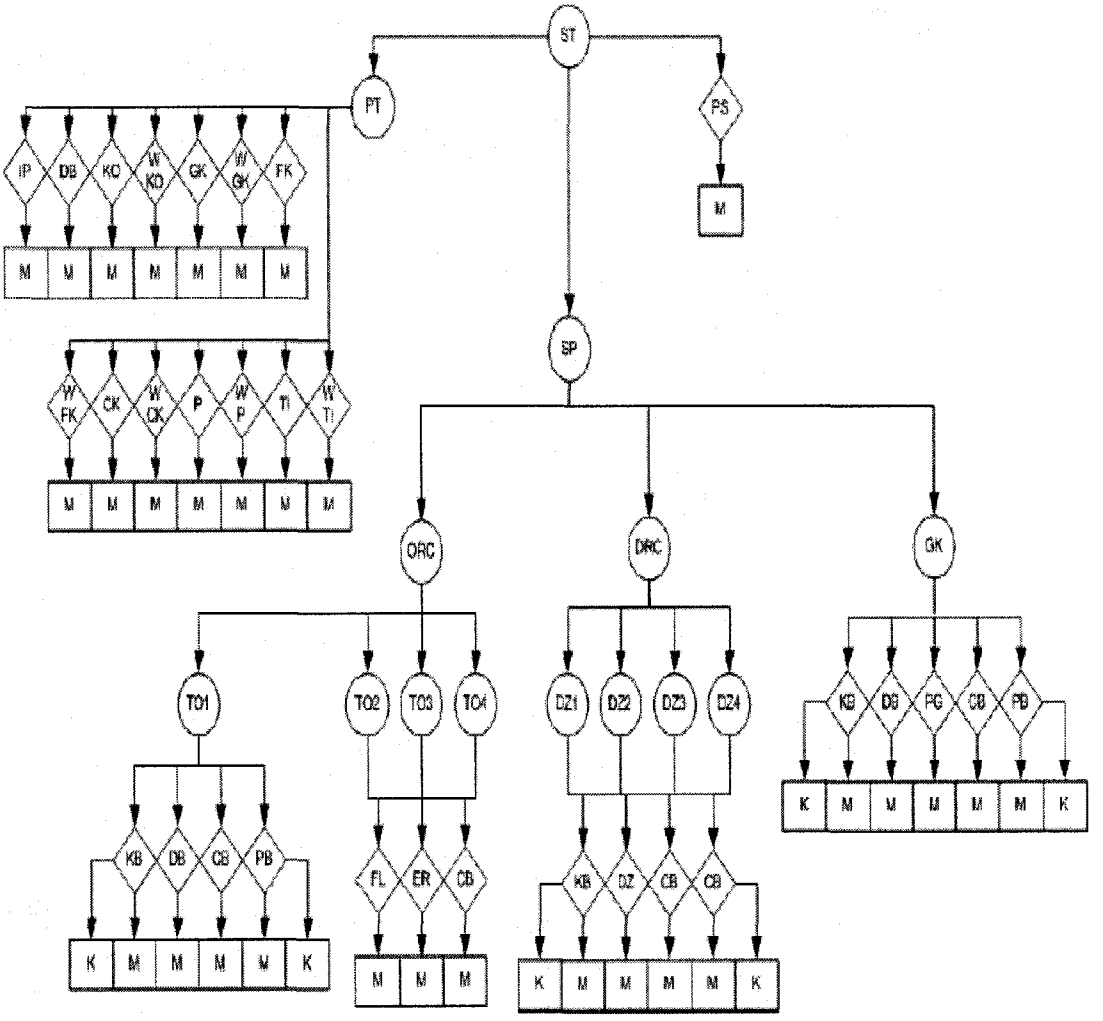


Table des Indexes

N

P

Q

R

S

T

V

Φ