Chapitre 4 : Application cible et co-simulation

Introduction

La propriété intellectuel développer NEURONA, nécessite une couche application qui supervisera le réseau de neurone, et jouera l'intermédiaire entre elle et l'environnement extérieur. Nous avons essayé de concevoir un environnement permettrant de co-simuler l'ensemble du system. Cette grande importance donner à la simulation et co- simulation découle (du chapitre I) du fait qu'il n'existe aucune méthode théorique pour trouver un réseau résolvant notre problème, aussi l'utilisation de la précision limitée risque de bloquer la convergence du processus d'apprentissage, le seul remède à ces limite théorique est la simulation exprèmental.

Nous allons étudier dans ce chapitre, le problème de la navigation du robot KHEPERA pour l'illustrer une méthode d'utilisation typique de l'utilisation de notre *IP*. L'environnement développé nous a permis, au moyen de la simulation, d'élaborer le choix adéquat des paramètres génériques d'implémentation hardware, ainsi que la co-simulation et la validation de l'interaction de NEURONA et du logiciel embarquer dans environnement simulé.

II. Environnement expérimental ciblé

L'application d'évitement d'obstacle peut être résolue en utilisant les réseaux de neurone. Nous disposons dans notre labo d'un robot kheperat et d'une carte FPGA Apex 2k de développement. Notre but est de crée une tourelle sur le robot portant une FPGA, qui prendra encharge les calculs neuronaux.

1 Le Robot Khepera

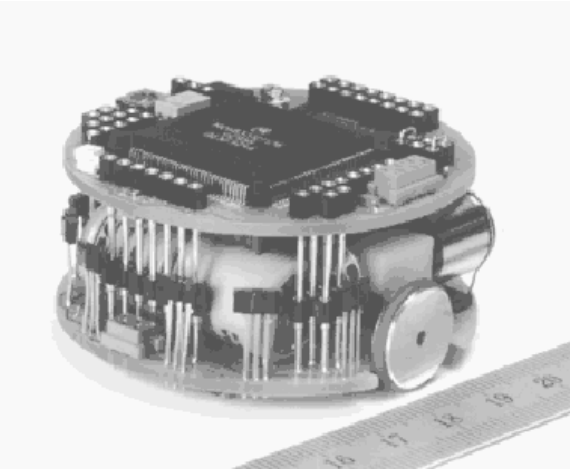

Figure 4-1 le robot Khepera

Khepera [15] est un robot miniature de forme circulaire destiné à la recherche (voir Figure). D'un diamètre de 55mm, d'une hauteur de 30mm, Khepera pèse 86g. Il a été développé afin de tester les algorithmes de contrôle en robotique dans le monde réel. Il est constitué de deux roues et 8 capteurs infrarouges (IR). Ces derniers permettent la détection des obstacles (émission/réception IR) et des sources lumineuses (réception IR). Six capteurs sont placés à l'avant du robot et les deux autres à l'arrière (voir Figure). Pour la mesure de distance, les capteurs renvoient des valeurs numériques comprises entre 0 (lorsque le robot est à une distance supérieure à 5cm de l'obstacle) et 1023 (lorsque le robot est à une distance inférieure à 2cm de l'obstacle). Pour la mesure d'intensité lumineuse, l'intervalle est de 50 (intensité forte) à 520 (intensité nulle).

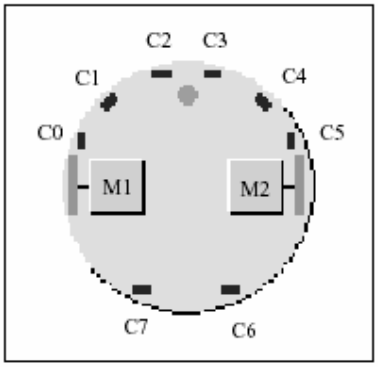

Figure 4-1 Emplacement des capteurs sur Khepera

La commande des moteurs entraînant les roues est donnée par des valeurs comprises dans un intervalle entre -10 (tout vitesse vers l'arrière) et +10 (tout vitesse vers l'avant). Les deux moteurs sont indépendants, ce qui permet de faire des rotations par l'intermédiaire d'une commande utilisant deux valeurs opposées (e.g. +5,-5 = réaliser une rotation en vitesse moyenne dans le sens horaire). Khepera peut être utilisé sur un bureau, connecté à une station de travail par un câble de liaison série du type RS232. Cette liaison permet de lire les valeurs des capteurs et d'envoyer des commandes au robot en temps réel. On peut aussi télécharger un programme de contrôle dans sa mémoire (Khepera est doté d'un processeur Motorola 68331) par la liaison série, puis de le déconnecter du câble série afin de pouvoir rester complètement autonome.

2. La tourelle FPGA

Différent développement de tourelle à base de FPGA ont été réaliser [kap], une est commercialiser par Applied AI Systems Inc., Canada [ops] ;

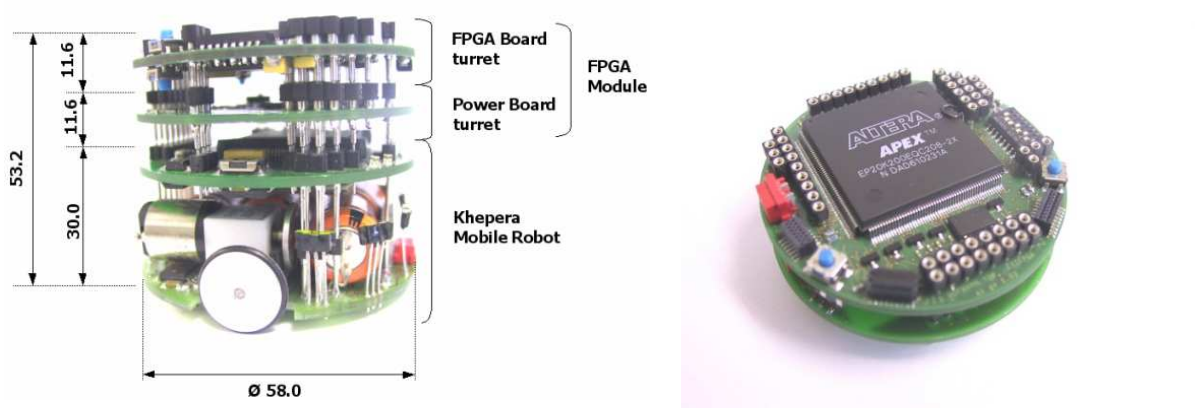

Figure 4-1 la tourelle FPGA

I. Cas d'étude : application d'évitement d'obstacle
1. Présentation générale

On présentant dans la figure 4-1 un modèle d'utilisation typique de processeur neuronal NEURONA, il met en jeux deux composantes du système neuronal embarqué : le modèle d'environnement et modèle d'application embarquée. Le environnement réalisé est spécifique à la navigation de robot mobiles, mais il peut être avantageusement transposé à d'autre application neuronal.

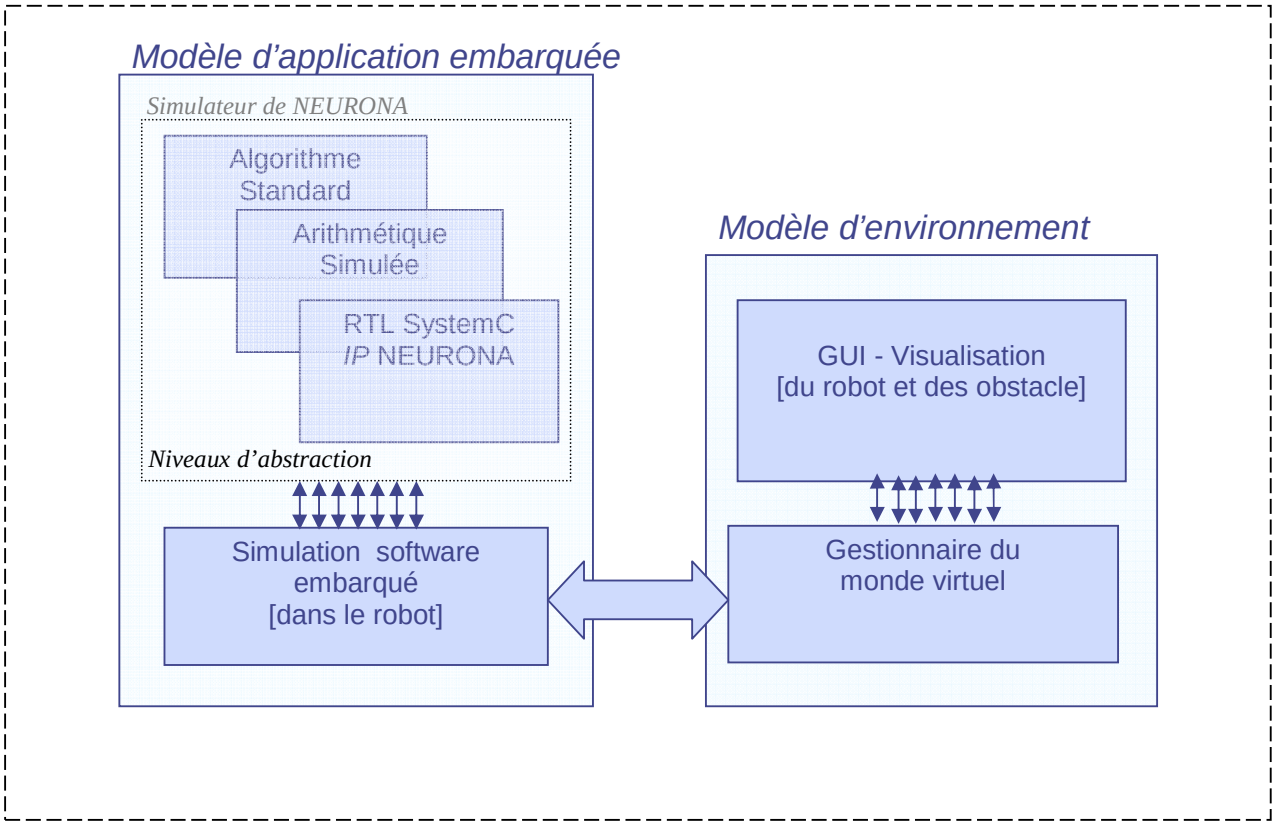

Figure 4-1 Prototype de co-simulation

Le model d'application embarquée :

L'application neuronale est composée du simulateur de NEURONA couplé à une couche embarquée chargé de la supervision de la navigation et du contrôle d'apprentissage. L'objectif principal de notre outil est d'aboutir à l'architecture optimale de NEURONA assurant le meilleur comportement. L'optimisation concerne le nombre de couche, le nombre de neurone et le dimensionnement de l'arithmétique Cette optimisation est obtenue en plusieurs itérations de

raffinement basé sur la simulation à divers niveaux d'abstraction que nous allons expliquer dans les paragraphes qui suive.

Au départ on utilise le niveau algorithmique standard utilisant la virgule flottante du système, En suite on teste la réduction de l'arithmétique par émulation de la virgule flottante puis de la virgule fixe, à l'issue de cette étape on s'assure d'avoir bien choisi les paramètres de l'arithmétique, En dernière étapes grâce à la Librairie SystemC, on optimise le model *RTL* synthétisable ceci par le moyen du model générique de NEURONA. L'intérêt de notre outil de validation est de régler les poids d'apprentissage en se basant sur le model RTL grâce à l'interopérabilité entre les divers niveaux d'abstraction permis, en utilisant un raffinement progressif en se rapprochant étape par étape vers les contraintes des matériels.

L'interaction entre NEURONA avec la supervision d'apprentissage est également prise en charge par l'outil. Ceci afin permettre de valider l'apprentissage en ligne et de tester l'auto adaptation de NEURONA à des situations imprévues.

Le model d'environnement :

La modélisation de l'environnement vise à créer un monde virtuel représentant le robot dans son univers afin de servir de test d'application neuronale embarquée.

Le monde virtuel est modélisé par les équations mathématiques décrivant la géométrie en 2D de l'espace de navigation muni d'obstacles ainsi que les équations d'émulation de comportement des capteurs et des actionneurs du robot.

L'interface GUI permet de créer et de modifier un environnement virtuel et de programmer et de visualiser des scénarios de navigation de manière conviviale et rapide « utilisé en guise de test bench ».

Afin de co-simuler le model hardware et software nous avons utilisé C++ et la librairie *SystemC* [9 Systemc] [8Systemc], ainsi que la librairie Qt [15Qt] pour la gestion des interfaces graphiques. L'usage conjoint de ces deux bibliothèques pose une difficulté technique une telle réalisation a été traitée par l'Université de Montréal [qtSyst] visant un simulateur SystemC via Qt). Le portage du logiciel d'apprentissage du l'environnement de co-simulation à environnement embarqué est facilité par le fait que tous les microcontrôleurs disposent d'un compilateur C.

L'environnement développé permet ainsi de valider le comportement global du système à l'aide de monde virtuel.

Conclution:

Nous essaierons dans les paragraphes qui suivront d'appliquer notre Flow de conception et de validation de IP, et nous essaierons de présenter la partie software embarqué.
Nous présenterons un simulateur développer, en premier lieu pour faire des simulations et voire le comportement du robot et la phase de l'apprentissage en réduisant la précision et en utilisant différentes arthritiques. Ensuite, le simulateur permet des co-simulation entre le module Hardware, le software embarqué le tous dans un environnent simulé ou on peut visualiser et agir sur l'environnement, les paramètres génériques, les données d'apprentissages vis une GUI. Ceci permet de voir directement l'effet des différents chois sur le comportement du robot.

2. Simulateur d'évitement d'obstacle

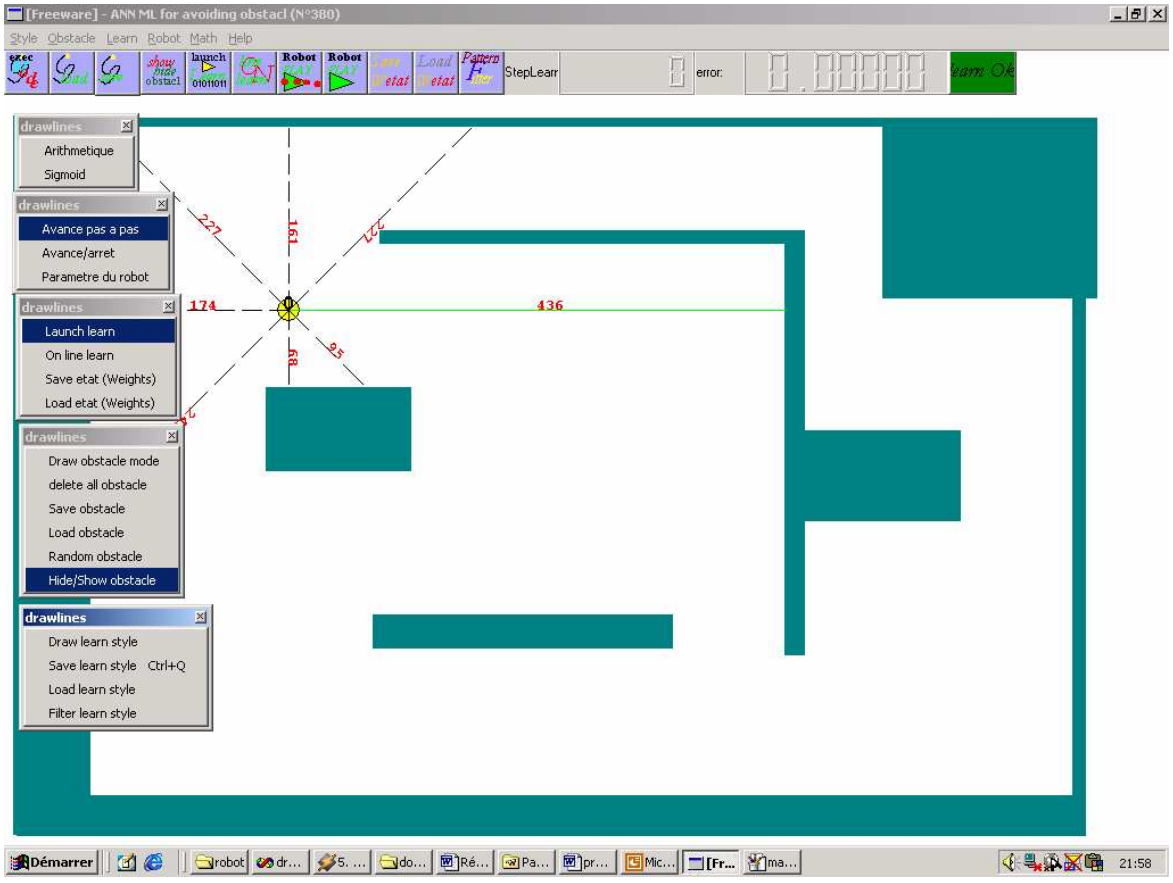

figure 4-2 Aperçu de l'application

Il serait très intéressant de valider et d'étudier les modules développés dans un environnement simulé avant de se lancer dans une application réelle.

En vue de créer cet environnement, on a du développer un simulateur de test. Ce dernier permet de gérer les paramètres de l'environnement (ici les obstacles) et les paramètres du robot : nombre de capteurs et leurs caractéristiques, angle de rotation maximal dans un cycle, les pas de déplacement à chaque cycle…

Le simulateur a été développé entièrement en C++. La librairie multi-plateforme *Qt* [15Qt] issue du monde libre a été retenue pour le développement de la partie graphique.
Une classe implémentant le réseau de neurones multicouches, a été développé, permettra de contrôler les réflexes automoteurs du robot. Cette classe peut utiliser une arithmétique variée en se basant sur les paramètres génériques du C++ (Templates). Deux arithmétiques ont été ainsi développées en exploitant la propriété de surcharge des opérateurs en C++ : la virgule fixe et la virgule flottante.

Un réseau de neurones choisi peut être instancier en utilisant indifféremment la virgule fixe ou la virgule flottante. Dans les deux cas il faut préciser le nombre de bits à utiliser : la mentisse et l'exposant dans le cas de la virgule flottante, le nombre de bits après et avant la virgule dans celui de la virgule fixe.

Pour mettre en évidence l'effet de la précision sur la fonction d'activation, l'application permet de choisir un type de précision pour la simulation. Deux types de précision sont disponibles : la précision réelle (utilisant les librairies mathématiques du langage C++) et la fonction approximée par intervalles. Les paramètres de l'approximation sont dans ce cas le pas et l'intervalle utilisé.

Les résultats de la simulation :

Les résultats de la simulation corroborent les résultats des recherches présentées dans le chapitre I. En effet, l'utilisation d'une précision réduite lors des différentes simulations n'a pas beaucoup influé le comportement du robot. Une arithmétique à virgule fixe utilisant 6 bits s'est avérée largement suffisante pour piloter le robot.

En revanche, dans le cas de la rétro-propagation, les simulations montrent la nécessité d'une grande précision afin d'aboutir à la convergence de l'algorithme. Vingt bits constituent un minimum et il n'est pas rare que l'erreur se bloque dans des paliers : l'algorithme n'arrive plus à converger.

L'étude expérimentale par simulation du système nous a permis d'avoir une idée assez nette des paramètres de l'implémentation hardware (nombre de bits, arithmétique, etc.) à développer et les performances matérielles attendues en termes de précision.

3. Développement du réseau de neurones.

a. Spécification des entrées/sorties.

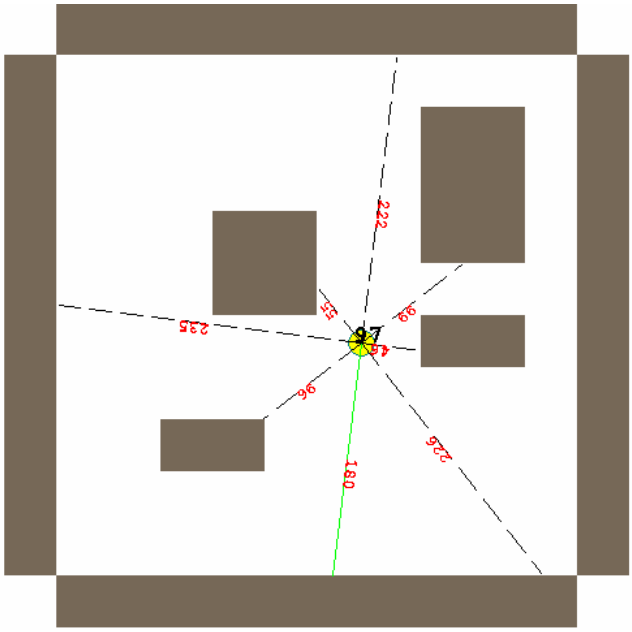

figure 4-3 Environnement du robot

Le robot doit décider de l'angle de rotation à faire en fonction des paramètres fournis par les capteurs. Ces paramètres sont les distances séparant le robot des obstacles suivant les axes des capteurs. On en déduit les caractéristiques du réseau de neurones à utiliser :

- Entrées : distances données par les différents capteurs.
- Sorties : Angle de la rotation à faire par le robot.

Le réseau approximera une fonction de $\mathbf{R}^n$ vers $\mathbf{R}$, où n présente le nombre de capteurs.

b. Collecte des données :

Notre simulateur permet à l'utilisateur de contrôler directement le robot, le trajet effectué par le robot est enregistré, et sera traduit en pattern pour l'apprentissage.

Cette méthode permet au robot d'imiter un comportement, un style, une stratégie d'évitement d'obstacle.

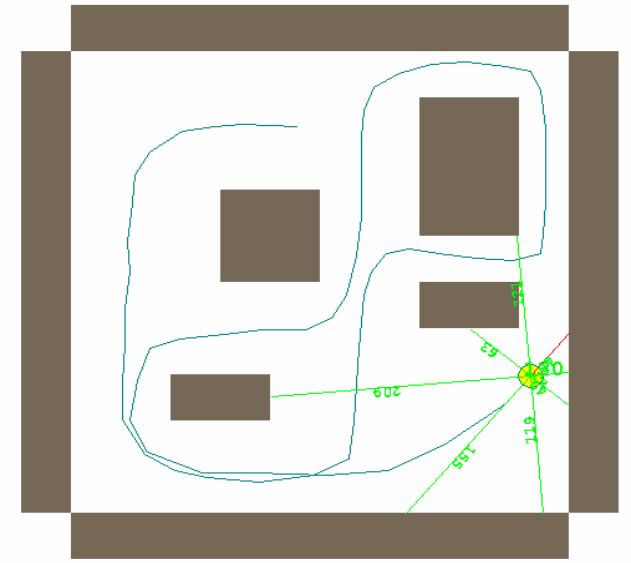

figure 4-4 Trajet apprentissage

c. Analyse des données

Les données collectées ne sont pas directement exploitables. En effet certains traitements sont nécessaires :

- il faut éliminer les données redondantes
- il faut éliminer les données contradictoires
- il faut simplifier la fonction à approximer pour accélérer la face d'apprentissage et éviter de la saturer. (Voir figure 4-5)

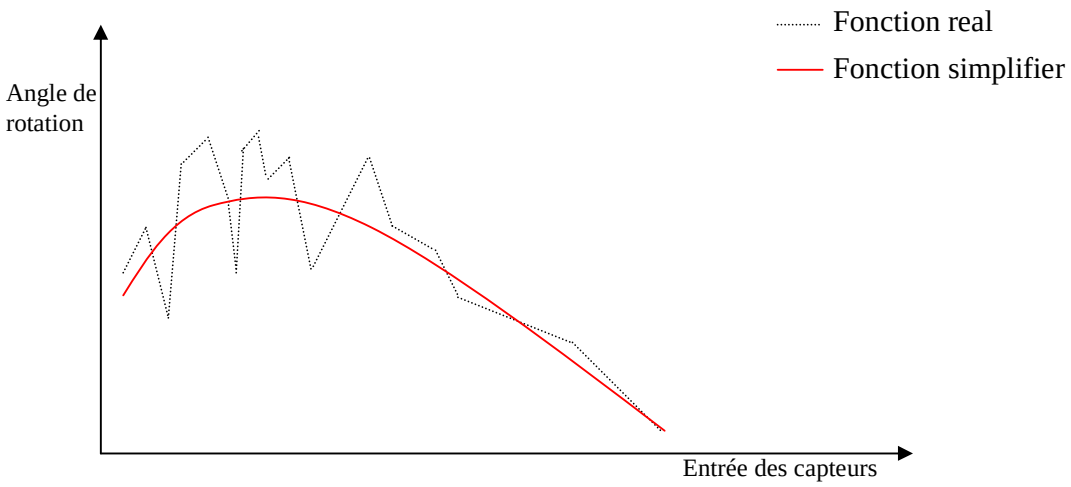

figure 4-5 simplification de la fonction a approximer

d. Mise en forme des données

L'angle de rotation varie entre α_{min} et α_{max}. Une bijection nous permettra de travailler dans [0,1].

Le champ d'action du capteur varie entre d_{min} et d_{max} . On transformera cet intervalle en [0,1] au moyen d'une bijection.

α_{min} et α_{max} sont fixées expérimentalement, on ne doit pas autoriser le robot a faire des mouvements brusques (Une série de tests expérimentaux nous a permis de fixer α_{min} et α_{max} à -30° et à +30° respectivement.).

d_{min} et d_{max} dépendent des caractéristiques des capteurs utilisés par le robot.

e. Choix du réseau

Plusieurs réseaux ont été explorés dans notre étude, les résultats expérimentaux réalisés montrent qu'un réseau à quatre couches suffit généralement à nos besoin.

f. Validation

Une fois l'apprentissage effectué, pour analyser la qualité et la capacité de généralisation du réseau utilisée,

En utilisant l'interface graphique, on peut construire et sauvegarder plusieurs environnements (monde), puis pour un monde choisi on peut lancer le robot.

Suivant le comportement du robot, la distance parcourue en une durée de temps fixe, l'espace couvert, habiliter à éviter les obstacles, on donne une note d'appréciation.

La qualité du réseau dépend du réseau utilisé, les vecteurs de test utilisès (style et le monde utilisée) et les environnements du test d'appréciation.

Pour améliorer les résultats du réseau on peut activer Apprentissage en ligne, qui intervient lorsque le superviseur juge que le robot appris une mauvaise décision, il rectifie en ajoutant un vecteur de test à la base de test utilisé pour l'apprentissage et relance l'algorithme d'apprentissage.

4. Apprentissage en ligne et superviseur.

L'apprentissage en ligne est supervisé par logiciel embarqué. Le réseau de neurone qui contrôlera les reflex du robot est initialement paramétré selon une stratégie d'imitation initiale choisi défaut.

L'évolution du robot dans un environnement inconnue peut engendrer des situations non couvertes par les patterns qui ont servi à l'apprentissage. Il arrive, toutefois, que le robot prenne des décisions erronées et ce en dépit de la propriété de généralisation des réseaux de neurones à partir d'un nombre réduit de cas. L'intervention du superviseur est nécessaire dans ce cas pour corriger la décision du robot. Le superviseur lance des cycles d'apprentissage avec des patterns qui rectifient les fausses décisions.

La détection correction d'une décisions erronées est un problème complexe, la solution adoptée consiste à attendre la collision du robot puis réétudier les trois dernières décisions avant la collusion en vue de générer un patterns de rectification. Ce pattern sera injecté dans la base d'apprentissage suivie d'une étape de filtrage, d'élimination d'incohérences, ensuite on relance du processus d'apprentissage.

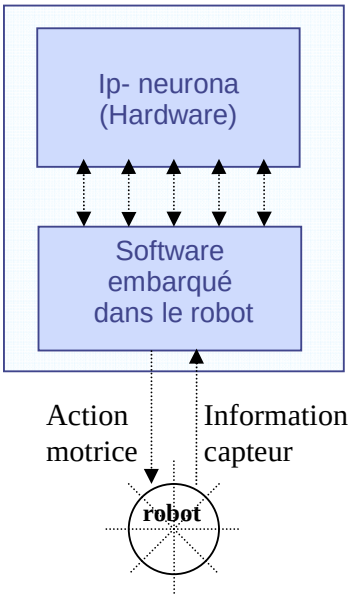

figure 4-6 System global

La supervision sera développée en tant que composante logicielle embarquée alors que tout l'aspect du calcul de l'apprentissage du réseau de neurone sera implémenté matériellement.

Le logiciel embarqué assure l'interface entre le module hardware et le monde extérieur (capteur & moteur) son rôle est la mise en forme des données entre hardware et embarqué, il pilote également l'apprentissage qui consiste à la présentation des vecteurs d'apprentissage un par un et à la décision d'arrêter ou continuer l'apprentissage.

5. Validation du composant matériel

La méthodologie traditionnelle de validation d'IP [FLowASIC], consiste à créer un générateur de pattern (vecteur de test) basé sur un algorithme de référence qui est implémenté en C++. Le fichier en sortie n'est généralement qu'un fichier *testbench* écrit en HDL, qui soumet une séquence de vecteurs de test en entrée et permet, après traitement, de vérifier la conformité des résultats obtenus avec les résultats prévus initialement.

Pour tous les niveaux de raffinement on a procédé par la même méthode :
Dans le cadre de ce projet, on adoptera une méthodologie légèrement différente. En effet, on compare les résultats d'exécution de l'algorithme de référence et de l´implémentation en SystemC. On exécute les deux algorithmes au sein d'un même programme ; des structures conditionnelles permettent d'assurer la comparaison.

Le déboguage se base essentiellement sur les points de comparaison pour trouver la source de l'erreur. On peut en ajouter, par exemple à chaque sortie de neurone ou encore pousser le raffinement plus loin en les plaçant avant et après le passage à la fonction d'activation.
Cette méthode nous permet de mesurer la différence entre les sorties de l'algorithme standard utilisant la virgule flottante ou fixe et notre implémentation en SystemC.

IV. Conclusion

La co-simulation représente une importante étape avant de passer dans le mode réel, elle permet de bien choisir les paramètres pour la partie hardware, et d'avoir une idée claire du comportement de l'ensemble dans un milieu bien choisi. Et d'ajuster la partie du l'algorithme embarqué suivant une métrique bien choisie.
Cette partie mérite d'être plus étudié, et ne pas figurer comme un cas d'application, elle traite un nouveau problème d'interaction hardware software, envoie le tous comme un seul module écrit en C++.
De même que la partie embarquée, il serait très intéressant de pousser plus loin l'étude et trouver d'autres algorithmes plus intelligents et plus adaptés à la diminution de précision.

[qtSyst] A Methodology for Interfacing Open Source SystemC with a Third Party Software , Luc Charest Michel, Reid E.Mostapha Aboulhamid Université de Montréal

[15Qt] Trolltech AS, Qt On-Line Reference Documentation: http://doc.trolltech.com.

[8Systemc] Synopsys.Inc, Coware Inc, Frontier Design.Inc ”SystemC User’s guide” ver 2.0

[9Systemc]Steve Holloway, David Long, Alan Fitch “from algorithm to SoC whith SystemC” Doulos ltd, année 2002

[FLowASIC] Santarini, Michael “ Million gate ASICS willrequire hierarchical flow” EE Times, année 2001

[kap] FPGA Module for the Khepera robot : Stéphane Hofmann Supervisors : Daniel Roggen and Yann Thoma

[ops] Evolvable Hardware Turret http://www.aai.ca/robots/khep_fpga.html

[15] K-TEAM S.A., Preverenges, Switzerland (http://www.kteam.com). Khepera User Manual.