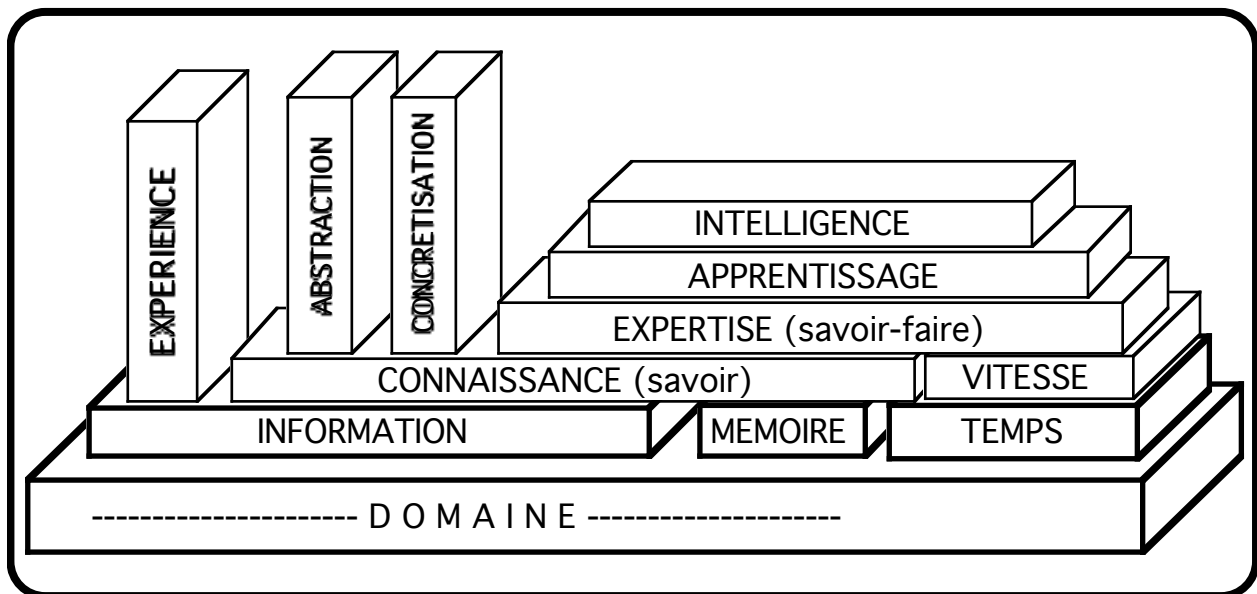


*Facultés de Technologies Industrielles (TIN), Informatique et Communication (TIC)
et Formation En Emploi (FEE)
Filières microtechniques, électronique et automatisation industrielle (MI, EAI, etc.)
LaRA - Laboratoire de Robotique et Automatisation*

INTELLIGENCE ARTIFICIELLE ET GENERATION DE PLANS D'ACTION

(Partie 4.6 du cours “Robotique et Automatisation”)

Jean-Daniel Dessimoz



INTELLIGENCE ARTIFICIELLE ET GENERATION DE PLANS D'ACTIONS**Table des matières**

1. Introduction.....	2
2. Bases théoriques pour les sciences cognitives.....	2
2.1 Modélisation – notion de modèle	4
2.2 Information.....	5
2.3 Cognition.....	6
2.4 Complexité.....	7
2.5 Connaissance	7
2.6 Expertise.....	9
2.7 Apprentissage / learning.....	10
2.8 Expérience.....	11
2.9 Intelligence.....	11
2.10 Abstraction et concrétisation.....	12
2.11. Mémoire.....	13
2.12 Complexité et réductibilité	14
3. Choix d'une structure de commande.....	16
4. Intelligence artificielle	19
4.1. L'intelligence artificielle	19
4.2. L'intelligence artificielle selon les canons traditionnels (« GOFAI »)	20
4.3 Langages de programmation en intelligence artificielle.....	21
4.4 ELIZA La machine "intelligente".....	24
4.5 Automates en situation ("insectes artificiels").....	35
4.6 Inférence Bayésienne	37
5. Logique floue.....	41
5.1 Présentation générale de la logique floue	41
5.2 Togai Infralogic - systèmes à logique floue	43
5.3 Réalisation HEIG-VD - système à logique floue pour la régulation d'un balancier.....	44
6. Réseaux neuronaux	45
6.1 Présentation générale des réseaux neuronaux.....	45
6.2 Réalisation HEIG-VD - système à réseau neuronal pour la régulation d'un balancier.....	47
6.3 Autres réseaux de neurones.....	58
7 Algorithmes génétiques.....	60
7.1 Définitions générales pour les algorithmes génétiques	60
7.2 Réalisation HEIG-VD - Algorithme génétique pour la régulation d'un pendule inversé	62
8. Robots coopératifs et humanoïdes.....	66
8.1 Robots mobiles autonomes robustes – Exemple Eurobot.....	66
8.2 Robots coopératifs à capacités cognitives développées – Exemple Robocup@Home	69
9. Autres exemples impliquant cognition et cognitive au LaRA	75
10. Commande multimodale	77
11. Robot humanoïde NAO	80
12. Conclusion.....	86
Références	86
Annexe - Archives - Systèmes-experts	87

INTELLIGENCE ARTIFICIELLE ET GENERATION DE PLANS D' ACTIONS

1. INTRODUCTION

Le terme d'intelligence artificielle recouvre des significations variées, et ce chapitre contribue à en préciser le sens. En premier lieu, et de façon vague, on attend de l'intelligence artificielle qu'elle contribue à automatiser la solution de problèmes complexes, qui sont encore peu ou mal résolus. Parmi ces problèmes, il se trouve en particulier la génération de plans d'actions. L'idée est ici que l'utilisateur désirant un certain résultat ne devrait pas avoir à spécifier toutes les actions élémentaires y amenant. Au contraire, il devrait n'avoir qu'à fixer l'objectif final. C'est ensuite le robot (ou plus généralement, la machine) qui devrait de lui-même (d'elle-même) en tirer le plan d'actions y amenant, ainsi bien sûr que de le mettre en oeuvre.

Il se trouve que les tâches de manutention et d'assemblage se révèlent volontiers complexes à définir de façon détaillée. Dans ce contexte, la génération automatique de plans d'actions est souhaitable. Le présent chapitre indique dans quelle mesure et comment les techniques de l'intelligence artificielle (I.A.) peuvent contribuer à résoudre ce problème.

Le chapitre est organisé de la façon suivante. D'abord, le domaine général de l'intelligence artificielle est brièvement présenté, ainsi que l'un de ses sous-domaines important: les systèmes-experts. Ensuite, divers aspects sont développés plus à fond: la partie 3 revient sur les systèmes-experts, le 4 discute les langages de programmation utilisés en I.A., et le point 5 présente des fondements théoriques pour la définition des concepts principaux de l'I.A. En particulier, les formules introduites permettent de vérifier que les modes traditionnels de traitement d'information (circuits électroniques, langages informatiques courants...) sont capables de très bonnes performances en termes cognitifs (quantité de connaissance, expertise, etc.).

Enfin le chapitre se termine en citant quelques domaines de l'I.A. (point 6), et en présentant quelques exemples: outils disponibles commercialement et systèmes à base de connaissance développés au laboratoire de robotique de l'HEIG-VD (point 7).

On ne peut traiter d'I.A. sans mentionner, outre certains domaines déjà mentionnés, quelques unes de ses spécialités.

Historiquement, on a vu systématiquement, en I.A., des domaines naître, acquérir une certaine maturité et s'en échapper: jeu d'échecs, vision par ordinateur, preuve de théorèmes mathématiques, traduction automatique...

Actuellement, certains spécialistes de l'I.A. traitent de physique qualitative, de modes temporels divers (passé, conditionnel, ...), de modèles multi-agents... Parmi les spécialités dont le retentissement est considérable, on mentionnera les réseaux neuronaux, la logique floue, et les automates en situation ("insectes artificiels").

2. BASES THEORIQUES POUR LES SCIENCES COGNITIVES

Le chapitre va aborder par la suite des domaines assez ciblés (réseaux neuronaux, probabilités bayésiennes, robots coopératifs, systèmes-experts, langages...). En fait, lorsqu'on prend la problématique de l'I.A. à sa base, c'est-à-dire au niveau des définitions (équations, unités...) il apparaît que le domaine scientifique généralement décrit comme relevant de l'I.A. ne se distingue fondamentalement ni par ses approches, ni par ses performances (ou alors par le fait qu'elles sont moindres...) de ce qui se fait ailleurs en génie informatique (aspects matériels et logiciels confondus). Le point 2 reprend plusieurs éléments de la publication [Des b].

Le domaine scientifique et technique de l'intelligence artificielle (I.A.) a commencé de s'explorer sous ce nom dans les années 50. Bien qu'une quantité considérable d'argent et de talent se soit consacré à ces activités, la simple définition des concepts clefs tels qu'intelligence, connaissance ou expertise ne s'est formulée clairement que récemment [Des a].

Par exemple, des références aussi réputées en I.A. que les livres de Nilsson ou de Michie ([NIL 82], [MIC 86]), ou aussi exhaustive que le manuel de Hunt pour l'I.A. ([HUN 86]) n'offrent pas de réponse satisfaisante. Dans le premier cas, le livre est complet au point d'inclure environ 300 références d'auteurs, et un index à 500 entrées. Mais cet index ne comprend pas de mots comme intelligence, expertise, ou connaissance comme tels; ces concepts ne sont pas définis formellement à l'intérieur du livre non plus. Dans le second cas, les mots intelligence et connaissance sont souvent mentionnés, mais les définitions formelles manquent aussi. Et dans le troisième exemple, comme cela est typique des dictionnaires, les définitions des mots intelligence et connaissances sont basés l'une sur l'autre (références circulaires). McCarthy ([HOF 90]) a aussi clairement fait apparaître combien une théorie axiomatique des faits et des valeurs cognitifs serait appropriée.

C'est le but de cette section de présenter des définitions pour les termes principaux en I.A., en construisant sur la base de la théorie de l'information. Les définitions sont en soi intéressantes. De plus, des équations permettent d'évaluer quantitativement les propriétés définies.

On a souvent proposé des métriques pour évaluer l'expertise ou le degré de connaissance. Mais ceci s'est fait de façon ad hoc, de façon adapté à certains domaines particuliers, mais sans saisir l'essentiel.

La théorie est applicable aux systèmes cognitifs de toute taille. Elle est applicable pour le cerveau humain (cf. test de Turing). Et elle peut aussi s'appliquer à un circuit électronique élémentaire (une "porte") ou pour une synapse unique. La théorie pourrait porter l'étiquette de "comportementale", comme les psychologues diraient, ou de "boîte noire" en jargon d'ingénieur des systèmes.

La section se découpe de la manière suivante: d'abord on rappelle les fondements de la théorie de l'information. Les parties suivantes introduisent successivement les concepts centraux en cognition: connaissance, expertise, apprentissage, expérience, intelligence et abstraction/concrétisation, tels que définis dans la théorie pour la cognition « MCS »..

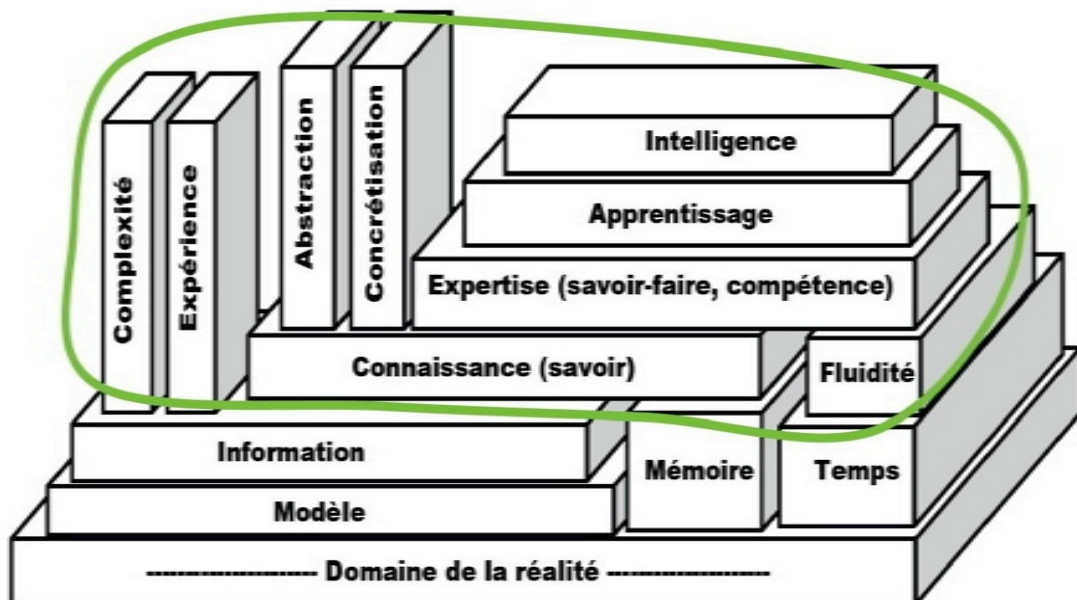


Fig.2.1 Représentation schématique de la façon selon laquelle les concepts cognitifs se construisent dans le modèle « MCS ». Les concepts hors ligne verte sont définis ailleurs et sont largement acceptés.

2.1 Modélisation – notion de modèle

Une étape de modélisation est toujours nécessaire pour focaliser l'attention sur les quelques aspects de la réalité pertinents pour l'un ou l'autre objectif d'intérêt courant. Ainsi implicitement d'innombrables propriétés de la réalité s'effacent et les processus cognitifs peuvent se développer.

Fig. 2.2

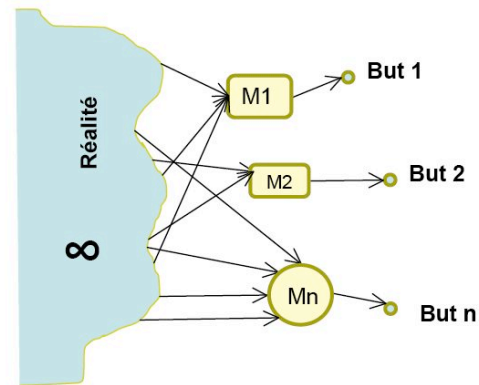


Fig.2.3 Il y a toujours une grande différence entre un objet réel (par exemple une pipe permettant réellement de fumer du tabac), et sa représentation picturale (ici le fameux tableau de Magritte) ; entre la réalité et n'importe lequel de ses modèles.

Un modèle, c'est généralement une représentation simplifiée de la réalité ; typiquement élaborée pour atteindre un certain but. Parfois, la correspondance avec la réalité n'est pas recherchée et il s'agit alors, par extension, de la représentation d'autres mondes, virtuels.

La correspondance entre modèle et réalité sous-jacente, définit la notion de sens.

Dans la mesure où un modèle permet d'atteindre un certain but, il peut se qualifier de bon pour ce but.

Les moyens de trouver un tel modèle sont divers et comprennent un ou plusieurs des éléments ci-dessous :

- essais et observation (expérience)
- élimination des paramètres d'importance non déterminante pour l'application
- intuition (brainstorming)
- raisonnement
- recours à un expert.

Paradoxe du modèle : « Un modèle, plus il est bon, plus il est faux ». Un modèle est bon s'il mène à l'objectif. Plus il est simple, plus il est bon. Or pour être simple, le modèle doit être allégé de tous les éléments non critiques pour atteindre l'objectif. Ce faisant il devient très incomplet et ne représente plus la réalité à laquelle il correspond. Dans ce sens, il est faux.

2.2 Information

Rappelons brièvement les définitions les plus fondamentales de la théorie de l'information. Pour une discussion plus complète, on se reportera au §1.3 *Capteurs et information*, ou à l'un des nombreux manuels consacrés à ce sujet.

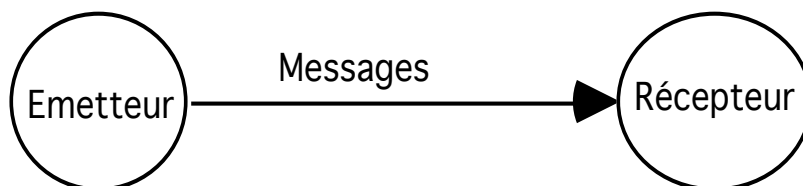


Fig. 2.4 L'information se transmet par messages.

Dans sa définition de base, l'information se transmet par des messages d'un émetteur à un récepteur, entre équipements techniques de communication (cf. Fig. 2.4).

Aujourd'hui, cette définition peut s'étendre à l'humain et aux systèmes cognitifs artificiels. Dès lors, le rôle de l'émetteur peut s'étendre à toutes sortes de sources, même totalement informelles, et par contre c'est le rôle du récepteur qui prend de l'ampleur et de l'intérêt.

L'information est une grandeur qui caractérise la surprise apportée par un message, sa nouveauté, ou, en termes statistiques, sa faible probabilité d'occurrence.

En bref, on peut dire que l'**information forme « l'opinion » du récepteur** ; elle lui permet de développer et mettre à jour son *modèle* de la réalité, dans un domaine particulier (cf. Fig. 2.5).

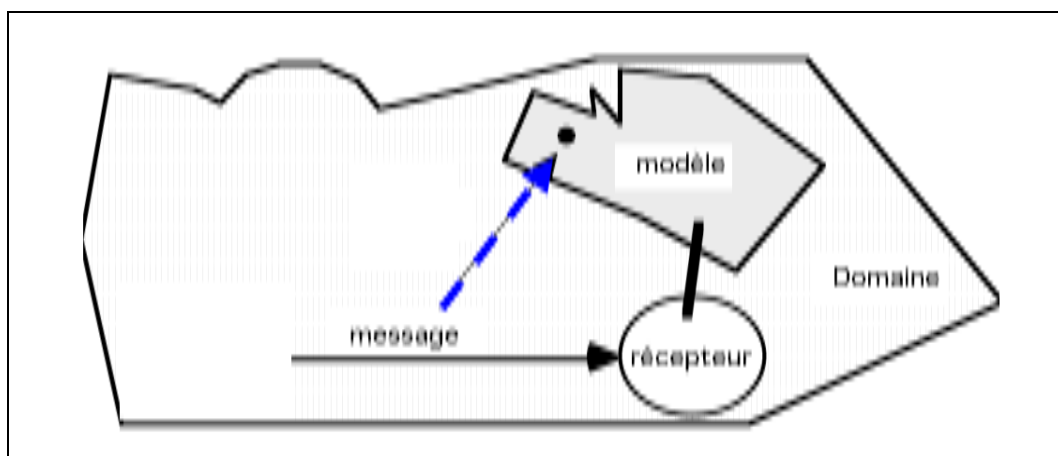


Fig. 2.5 Les messages ont un sens par référence au modèle interne qu'un récepteur a d'un domaine donné.

Avant réception du message, le récepteur a certaines attentes à son sujet. Ceci peut s'évaluer en termes de probabilité subjective, c.-à-d. comme la probabilité d'occurrence que le modèle attribue (peut-être de façon implicite) à ce message singulier.

La quantité d'information contenue dans un message de probabilité d'occurrence p :

$$Q = \log_2 (1/p) = -\log_2 p \text{ [bit]} \quad (1)$$

En conséquence, la quantité moyenne d'information délivrée par de tels messages sur un domaine D (n événements possibles, de probabilité individuelle p_i), est la suivante:

$$Q_{\text{moyen}} = -\sum p_i \log_2 p_i \text{ [bit]} \quad (2)$$

L'application de cette équation 2 aux messages entrant dans un système, ou en sortant, nous donnera les quantités n_i , n_0 , and n_{0S} dont on parle dans la suite.

Notons que les signaux continus, analogiques, peuvent toujours en pratique se convertir sans pertes sous forme discrète, numérique, notamment par échantillonnage et quantification opérés dans les règles ($f_e > 2 f_{max}$, et $N_v > S/Br$). Dès lors, les équations ci-dessus sont aussi applicables pour ces cas a priori non-discrets.

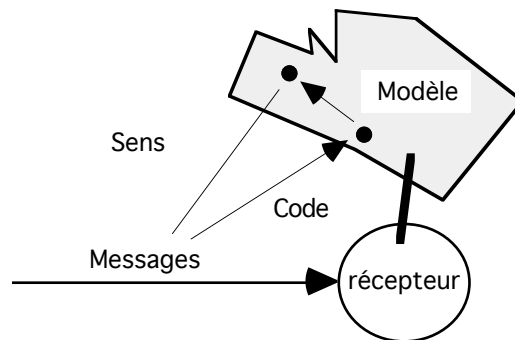


Fig. 2.6 La notion de code apparaît dans les cas où le message prend son sens de façon indirecte, c.-à-d. désigne l'élément cible du modèle via un ou plusieurs autres éléments spécifiques du modèle.

Attention aux deux faits troublants suivants : la quantité d'information est par définition subjective, et par nature, elle évolue au cours du temps, car elle a précisément pour mission de modifier les attentes du récepteur, de changer son opinion (modèle).

2.3 Cognition

La cognition est un terme scientifique et technique qui décrit d'une part la capacité générale d'un système à disposer de connaissances, c'est-à-dire à générer les informations pertinentes dans un domaine donné, et qui d'autre part concerne l'ensemble des processus y relatifs.

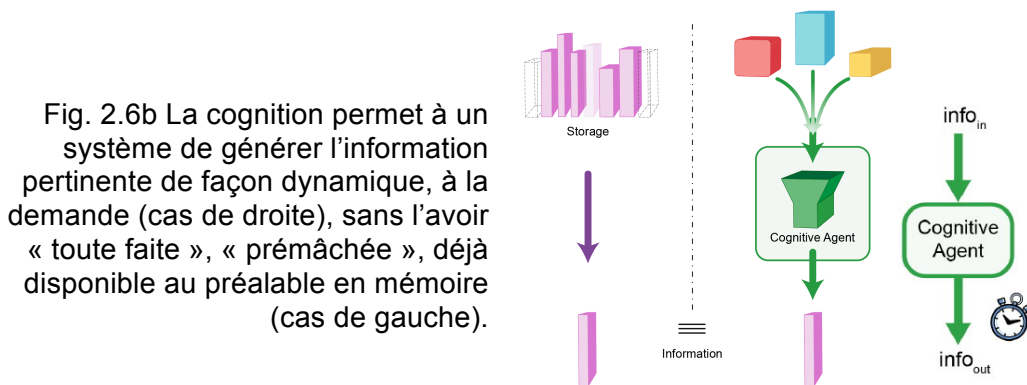


Fig. 2.6b La cognition permet à un système de générer l'information pertinente de façon dynamique, à la demande (cas de droite), sans l'avoir « toute faite », « prémâchée », déjà disponible au préalable en mémoire (cas de gauche).

Dans la théorie pour la cognition « MCS », la cognition opère dans des agents. Il s'agit d'un modèle comportemental, ou « boîte noire », se caractérisant essentiellement par les flux d'information en entrée et en sortie; le modèle est applicable à différents niveaux de détails.

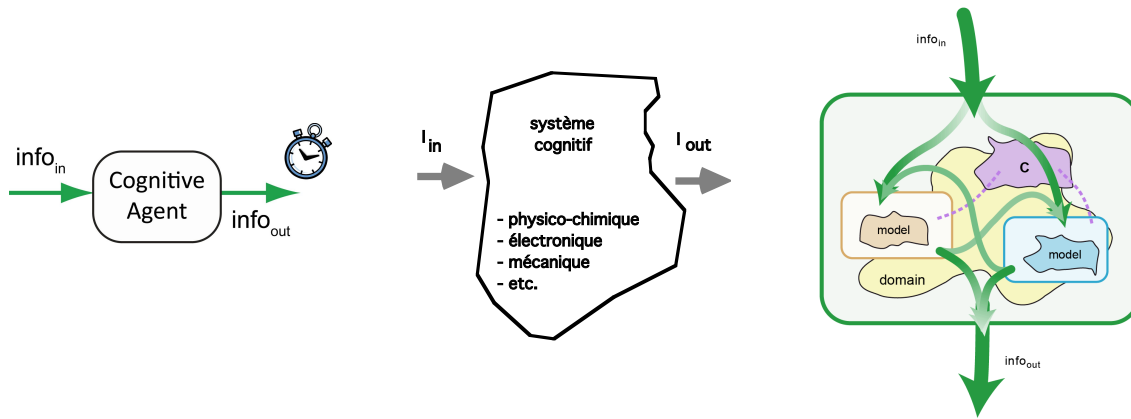


Fig. 2.6c Modèle comportemental (à gauche), implémentable de diverses façons concrète (centre), et à différentes échelles : individu mais aussi groupe ou sous-systèmes (droite)

2.4 Complexité

La complexité, c'est la propriété d'un objet qui requiert une grande quantité d'information pour être décrit de façon exhaustive. L'unité de mesure est celle de l'information, c'est-à-dire le [bit] .

2.5 Connaissance

Voyons la définition scientifique et technique du concept "connaissance".

La connaissance est une caractéristique éventuelle d'un *procédé*. Par extension, elle s'applique aussi au système qui implémente le procédé. Contrairement à l'information, qui est statique et prend typiquement l'aspect d'un ensemble de symboles (signes conventionnels), la connaissance implique par nature une évolution au cours du temps.

La connaissance, c'est la propriété qui fait qu'un système génère de l'information pertinente, typiquement à partir d'information incidente. Le domaine de connaissance peut se représenter comme un ensemble d'associations entre informations "d'entrée" et "de sortie", entre données et résultats. De plus ces associations sont orientées par rapport au temps. On les qualifiera de temporales. L'information de sortie est dite ici pertinente si elle correspond au domaine de connaissance.

On pourrait être tenté de faire l'économie du concept de connaissance et s'en tenir à celui d'information (en fait, ceci explique probablement pourquoi pendant longtemps la connaissance n'a pas reçu d'assise technique en propre): tous les messages de sortie voulus pourraient être tabulés en fonction de chaque message possible en entrée. Malheureusement on se heurterait, dans la plupart des cas, à des quantités prohibitivement élevées d'information à gérer. Une telle approche peut néanmoins utilement servir de référence.

La propriété de connaissance a l'aspect fascinant de permettre à un système de fonctionner comme s'il avait une quantité énorme d'information en stock, alors même qu'il exploite en fait une cascade (il serait mieux de parler de "réseau" si le terme n'avait pas tant de connotations) d'associations simples, elles-mêmes en nombre réduit .

Mathématiquement, la quantité de connaissances, K , existant sur un domaine D_K (l'ensemble de toutes les associations temporales entre données et résultats) se définit par l'équation suivante:

$$K = \log_2 (L+1) \text{ [lin] } \text{ Logarithme de l'INformation} \quad (3)$$

où L est la taille d'une table de conversion virtuelle (LUT) qui associe à tout message possible en entrée d_{ij} , la sortie correspondante, d_{oj} . Le terme "1" empêche m de devenir

négatif dans le cas où L prendrait de très petites valeurs. En posant n_i [bit] l'information moyenne contenue dans les messages d_j et n_0 [bit], celle de d_0 , nous avons:

$$L = n_0 * 2^{n_i} \text{ [bit]} \quad (4)$$

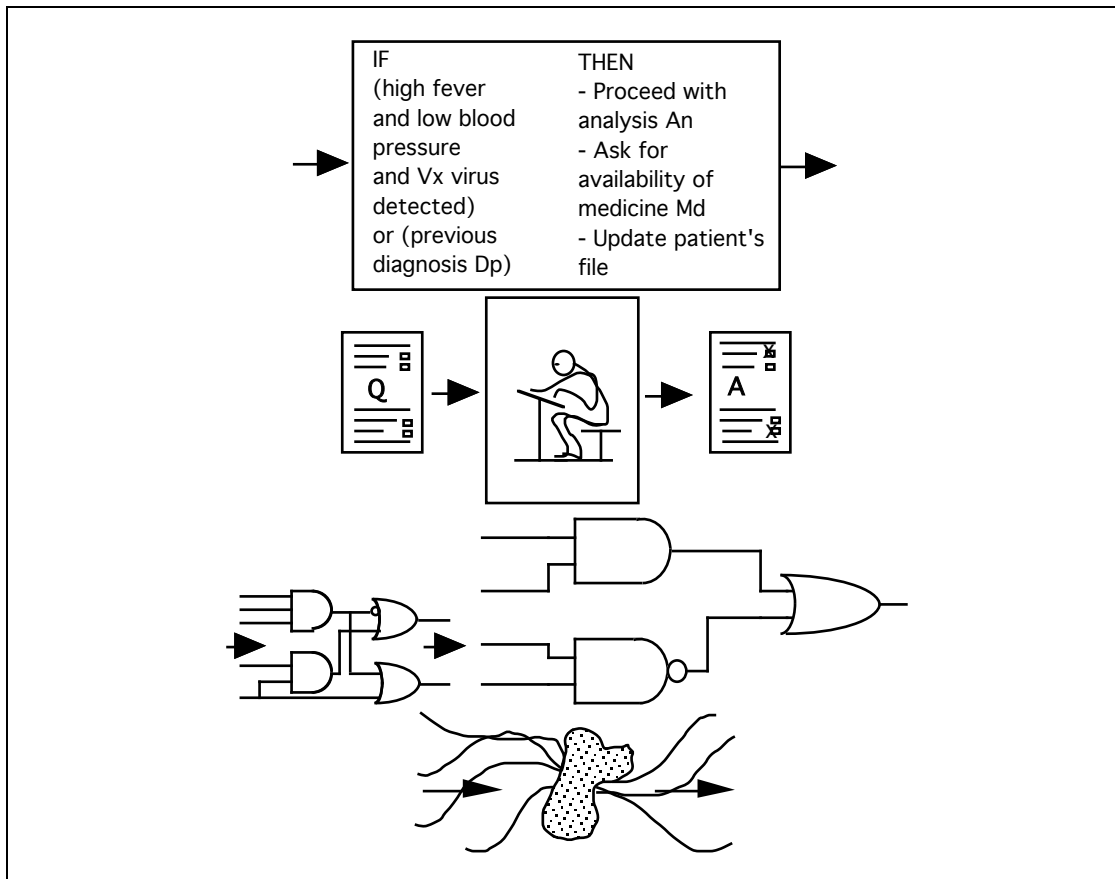


Fig.2.7 Exemples de systèmes caractérisables en termes de connaissances (règle de production, être humain, circuit électronique, cellules cérébrales)

En général, un système doté de connaissance ne contient pas en mémoire la table virtuelle car sa taille, L , est "énorme". La nature de la grandeur appelée "connaissance" est différente de celle de l'information, et cette seule raison suffit à justifier une nouvelle unité. Mais en plus, le fait que la taille de la table équivalente est gigantesque conduit à proposer une comme fonction de compression, le logarithme dans (3).

Remarquons que, comme ici définie, la connaissance n'implique aucune sorte particulière de structure interne. Elle est purement liée à la capacité du système à délivrer des résultats correspondant aux données entrantes. En particulier, la solution impliquant une table-mémoire est acceptable du point de vue théorique. Bien que la table ne contienne que de l'information, le système complet, avec sa faculté de gestion de la table (accès aux adresses appropriées, et livraison du contenu) doit être décrit par une autre propriété, qui est précisément celle de "connaissance".

Il peut arriver qu'un système ne connaisse pas parfaitement le domaine concerné D_m . Dans ce cas, ses messages de sortie, d_{0sj} , ne correspondront pas exactement aux associations du domaine, qui, elles, donnent d_{0j} . L'équation (4) est toujours valable, mais il faut déduire de l'information délivrée par le système ce qui ne correspond pas au domaine, et que l'on pourrait appeler "bruit", ou plus sévèrement "erreur". La quantité d'information correcte délivrée par le système, n_{0sc} , doit alors être évaluée, puis injectée dans la formule (4):

$$L_s = n_{0sc} * 2^{n_i} \text{ [bit]} \quad (5)$$

Nous pouvons exprimer n_{osc} de la façon suivante:

$$n_{osc} = - \sum_{i=1}^n p(d_{os_i}) p(d_{os_i}=d_{o_i}) \log_2 p(d_{os_i}) \quad (6)$$

où $p(d_{os_i})$ est la probabilité d'occurrence du message d_{os_i} sortant du système, et d_{o_i} est le résultat correct, c.-à-d. le résultat du domaine considéré correspondant à d_{ij} . Le terme $p(d_{os_i}=d_{o_i})$ est la probabilité du $i^{\text{ème}}$ message de sortie du système d'être correct. Remarquons que la condition d'égalité ($d_{os_i}=d_{o_i}$) est plus restrictive que la simple cooccurrence ($d_{os_i} \cap d_{o_i}$) habituelle en probabilités conditionnelles. l'idée fondamentale est ici que la quantité d'information livrée par chaque message de sortie du système devrait être pondérée par sa probabilité d'être correct. Si les réponses du systèmes sont en fait toutes correctes, de deuxième terme dans l'expression de droite de (6) a un effet nul (facteur égal à 1) et en conséquence les trois grandeurs n_o , n_{os} , and n_{osc} seront identiques. Dans le cas extrême inverse, si les messages de sortie ne correspondent pas au domaine de connaissance, ou, pour être bref, les réponses sont incorrectes, n_{osc} devient égal à zéro, ce qui conduit à 0 [lin] de connaissance, même si n_{os} est bien plus grand que n_o (cas d'un système... "bavard").

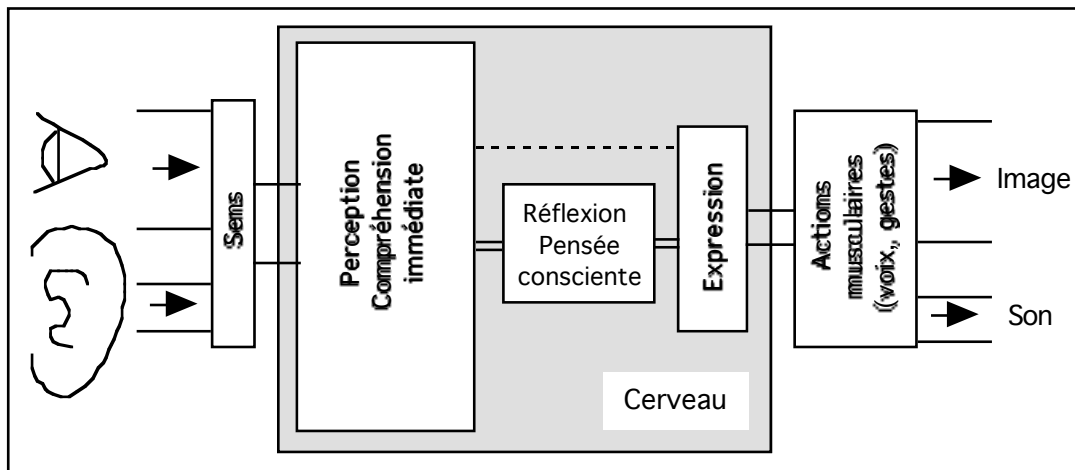


Fig.2.8 Canaux cognitifs de l'homme pour l'audio-visuel

2.6 Expertise

La connaissance est une propriété ou une caractéristique liée à un procédé traitant de l'information. Mais ce procédé peut se caractériser encore par d'autres grandeurs. Outre les domaines des messages d'entrée et de sortie, c'est encore la cadence de traitement qui lui est essentiel.

De façon surprenante, il est souvent facile de créer des systèmes contenant de très grandes quantités de connaissances, telles que des milliers voire des millions de [lin] (ceci est encore plus surprenant lorsqu'on se souvient que l'on parle du logarithme de la quantité d'information contenue dans une table "équivalente"). Considérons par exemple la quantité de connaissances nécessaires pour réaliser l'addition de deux nombres à 1 million de chiffres.

En pratique, ce qui importe aussi, c'est la vitesse avec laquelle le système traite l'information. La conjonction de connaissances et de vitesse de traitement se traduit en français par le mot expertise.

Formellement, l'expertise se définit ainsi:

$$E = K c \quad [\text{lin/s}] \quad (7)$$

où

- K est la quantité de connaissance du procédé, mesuré en [lin], et
- c, la cadence moyenne de traitement; précisément, "c" est l'inverse du temps de cycle moyen, c.-à-d. du temps qui sépare le message de sortie du message d'entrée; il se mesure en [1/s].

Cette équation met en lumière les limites de bien des ... systèmes savants (c.-à-d. dotés de connaissances). En fait, des quantités respectables de connaissances se heurtent, dans cette équation, à de très mauvaises performances en vitesse de traitement. Le résultat est alors assez quelconque en termes d'expertise, et des systèmes rapides possédant peu de connaissances peuvent facilement en faire autant.

Remarquons ici que l'équation (7) permet différents compromis pour un système. A quantité d'expertise égale, il est peut-être possible d'augmenter la cadence de travail, quitte à diminuer quelque peu la quantité d'information correcte en sortie, ou réciproquement.

Si L_S , plutôt que L, s'introduit dans (3), l'équation (7) a aussi la propriété désirable de garder l'estimation probablement constante si un domaine de connaissance plus grand, couvrant des champs encore inconnus du système, est pris en considération.

2.7 Apprentissage / learning

L'apprentissage, c'est la faculté que possède un système d'augmenter son expertise, c'est-à-dire sa quantité de connaissances ou sa cadence de travail. Un tel système peut apprendre par observation "passive" du domaine de connaissances, en l'explorant de façon organisée ou en recourant à l'aide d'un instructeur.

De façon qualitative, la capacité d'apprendre est démontrée si au temps t_1 , plus grand que t_0 (ou, comme on le verra plus tard, pour une expérience r_1 plus grande que r_0) le niveau d'expertise, tel que donné par (6), est plus grand.

Quantitativement, Q_I , c.-à-d. combien a été appris, s'exprime en termes d'expertise, c.-à-d. en [lin/s]:

$$Q_I = E(t_1) - E(t_0) = E(r_1) - E(r_0) \quad [\text{lin/s}] \quad (8)$$

2.7.1 Auto-apprentissage "passif" (AAP)

Dans le monde réel, le cas le plus courant est probablement celui où l'on apprend par l'observation tout à fait passive de phénomènes. Les associations données-résultats se succèdent de façon spontanées. Ce mode offre l'avantage de n'imposer aucun schéma de connaissance préétabli. Par contre, il est relativement lent car au début toutes les structures doivent être élaborées, et, après un certain temps, le taux d'associations instructives parmi l'ensemble des associations incidentes devient faible.

Ce premier mode d'apprentissage sert de référence pour estimer l'indice d'intelligence défini ci-dessous.

2.7.2 Auto-apprentissage "actif" (AAA)

Si un système doit acquérir de lui-même les connaissances d'un domaine qui lui est entièrement nouveau, le mode passif (AAP) décrit ci-dessus est probablement un bon point de départ. Cependant lorsque la plus grande part des connaissances sont acquises, (c.-à-d. que les structures internes principales sont construites), il devient plus efficace, si possible, de susciter les associations instructives, c.-à-d. les associations qui ne sont pas encore connues de façon ferme, ou qui puissent valider les modèles internes.

En mode d'auto-apprentissage "actif" (AAA), un système peut se voir comme réalisant de lui-même les expériences supposées conduire à des résultats instructifs. On ne discutera pas ici des stratégies appropriées en mode AAA (stimulation Monte-Carlo, essais-et-erreurs...) ni de la façon selon laquelle l'apprentissage se réalise en fait.

2.7.3 Apprentissage guidé

Les avantages et défauts du mode d'apprentissage guidé (AG) sont complémentaires à ceux du mode AAP. Il faut ajouter qu'il est essentiel ici de disposer d'un guide.

Cependant, il n'est pas toujours possible de trouver un instructeur, comme dans le cas de nouveaux domaines de connaissances ou pour des changeants où les changements sont rapides. Pour cette raison, le mode AG n'est pas toujours applicable.

2.8 Expérience

L'expérience a en français deux sens bien différents. Dans le premier cas, le mot correspond en quelque sorte à l'habitude que l'on a d'un milieu donné ("EH" pour expérience-habitude, correspondant au mot anglais *experience*), alors que dans le deuxième, le mot traduit la notion ponctuelle d'essai ("EE" pour expérience-essai, en anglais: *experiment*).

Les deux types d'expérience sont définis tour-à-tour ci-dessous. L'expérience (EH) se mesure habituellement en temps. On propose ici de l'évaluer en termes de nombre d'essais (EE) relatifs à un domaine de connaissances donné.

2.8.1 Expérience au sens d'essai

Pour apprendre, un système doit recevoir de l'information. Notre définition de l'expérience-essai, EE, est tout-à-fait compatible avec le sens courant d'opération exploratoire (manipulation de laboratoire par exemple).

L'expérience EE est une procédure qui apporte une ou plusieurs associations temporales, c.-à-d. un ou plusieurs couples données-résultats. Ce qu'on obtient, c'est de l'information pure, et cela se quantifie en [bit].

Dans le langage courant, le mot "expérience" (au sens EE) a une connotation technique. Elle doit néanmoins se comprendre ici de façon plus universelle. Par exemple, dans le jargon philosophique, il pourrait remplacer le mot phénomène; ou l'expression "étude de cas" en gestion d'entreprise.

2.8.2 Expérience avec notion de durée

L'expérience EH se comprend communément en termes de durée dans un environnement donné (par exemple: expérience d'une année en programmation Ada). Nous suggérons ici qu'il serait plus approprié de l'évaluer en termes d'exposition à un certain nombre d'essais (EE) sur un certain domaine.

Plutôt que de mesurer l'expérience (EH) en unités de temps [s], il serait préférable de la considérer comme une somme d'expérience au sens EE, et donc de l'évaluer également en unités d'information [bit].

La différence entre la mesure traditionnelle et cette nouvelle proposition peut paraître quelque peu académique, car en pratique, on caractérisera souvent un environnement particulier par le débit moyen d'associations qu'il génère dans un domaine de connaissance donné: dans ce dernier cas, le produit du débit d'EE par le temps d'apprentissage nous ramène au [bit].

2.9 Intelligence

Dans le cadre des présentes définitions, il est simple de définir précisément l'intelligence. L'intelligence, c'est la propriété qui rend un système cognitif capable d'apprentissage¹.

Dans la théorie MCS, l'intelligence peut s'estimer quantitativement comme un indice, comme le rapport entre quantité apprise, Q_L , ou simplement L , et expérience, R .

De la même façon que deux formes d'expérience ont été définies – l'une plus conventionnelle et intuitive, par rapport au temps, et l'autre plus rigoureuse, par rapport à l'information observée -, nous avons ici deux formes pour l'indice d'intelligence :

¹ La théorie MCS vise à la clarté et à l'essentiel ; de façon générale, aussi bien que dans le cas particulier de la présente définition. Il faut néanmoins reconnaître que dans le langage courant, le mot intelligence est plus ambigu, recouvrant parfois aussi divers autres sens, tels que « information », « connaissance », « renseignement » ou « compréhension ».

$$i_{l_i} = \frac{L}{R_T} \left[\frac{\text{lin}}{s^2} \right] \quad (9)$$

$$i_{l_i} = \frac{L}{R_i} \left[\frac{\text{lin}}{s \cdot \text{bit}} \right] \quad (10)$$

Pour un domaine donné, l'intelligence peut se quantifier essentiellement en prenant la dérivée à l'origine, p (pente initiale), de l'expertise E en fonction du nombre d'associations expérimentées, r :

$$i = dE/dr = p [\text{lin}/(\text{bit} \cdot s)]$$

Il est préférable d'amener le facteur d'intelligence dans l'intervalle entre 0 et 1; on peut alors définir un facteur d'intelligence normalisé, i_n :

$$i_n = 1 - 1/(1+i)$$

La pente diminue à mesure que l'expertise augmente parce que les associations rencontrées sont de moins en moins nouvelles pour le système.

La fonction $E=f(r)$ est généralement une grandeur statistique, et y mesurer une pente peut s'avérer délicat, particulièrement lorsque le domaine est petit. Néanmoins le principe est toujours valable.

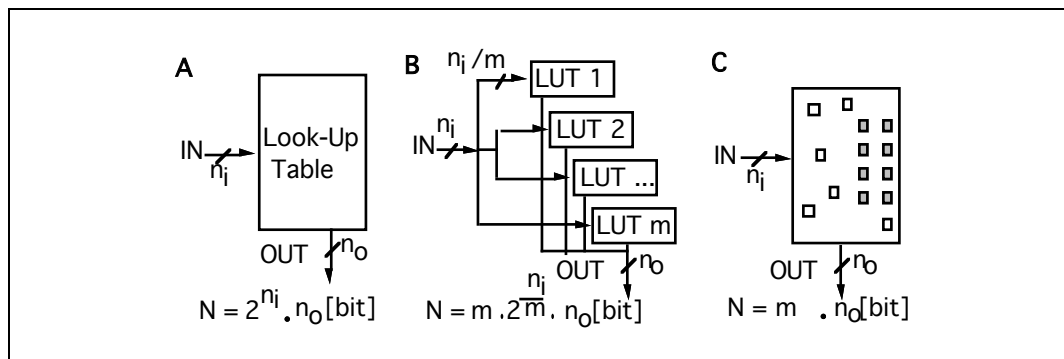


Fig 2.9. **Exemples de structure interne pour un système de connaissance (1/2).** A: Référence pour la définition du concept de connaissance. C'est peut-être aussi la réalisation la plus rapide, si elle est faisable (p. ex. multiplication par 3, ou fonction de transfert pour canaux video). B: Si les messages d'entrée peuvent être découpés en segments indépendants, la taille effective de la mémoire peut être plus petite que la LUT équivalente, sans perte de performances (p. ex. approche du "diviser pour régner", canaux RVB). C: Les besoins en termes de quantité de mémoire peuvent être diminués par une implémentation partielle de la LUT de référence (liste de termes logiques, jeu de règles de production, PLA,...). En général cela ne permet malheureusement pas une connaissance totale du domaine, mais c'est peut-être néanmoins l'approche la plus courante dans le monde réel.

2.10 Abstraction et concrétisation

L'abstraction, c'est la caractéristique d'un système dont la quantité d'information en sortie est plus petite qu'en entrée. Lorsqu'au contraire, les messages délivrés en sortie contiennent une information pertinente en quantité plus grande qu'en entrée, il y a concrétisation.

Chez l'homme par exemple, les procédés d'abstraction sont étroitement associés à la perception, alors que les procédés de concrétisation sont typiques des phénomènes vocaux et moteurs. Les deux types de procédés peuvent se décrire avec exactement les mêmes formules pour la quantité de connaissance, l'expertise, etc..

Le même indice pourrait aussi servir à quantifier le niveau d'abstraction et celui de concrétisation:

$$i_{AC} = n_o / n_i$$

Lorsque i_{AC} est plus petit que 1, il s'agit d'abstraction, alors que dans le cas contraire, c'est de concrétisation qu'il s'agit.

Cet indice est particulièrement satisfaisant dans le cas de la concrétisation. Pour l'abstraction cependant, il est peut-être plus agréable de travailler avec un indice qui atteigne des valeurs élevées lorsque le niveau d'abstraction est élevé. Introduisons i_A :

$$i_A = n_i / n_o$$

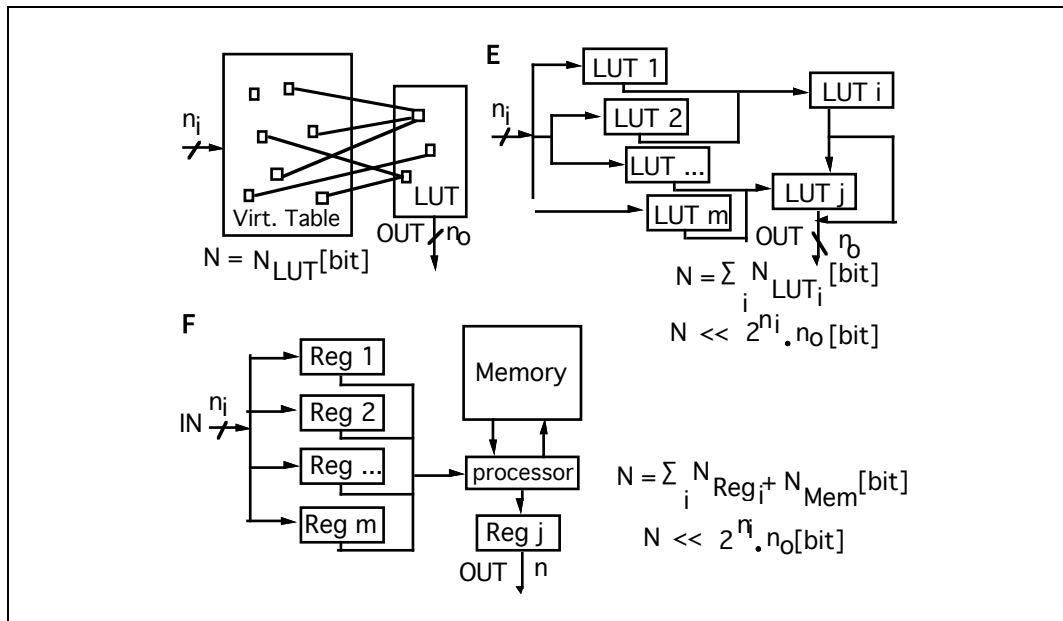


Fig 2.10. Exemples de structure interne pour un système de connaissance.(2/2)

D: Voici un cas où une projection extrêmement efficace et spécialisée se produit, qui conduit à un domaine de taille plus petite et de généralité plus grande (p. ex. transformation de Fourier, dans l'oreille, ou extraction de contours, dans l'oeil. Bien des types intéressants de projections ont été définis en sciences). E: Les résultats s'obtiennent par une cascade de LUTs de taille relativement petite (p.ex. additionneur arithmétique). F: Représentation schématique d'un ordinateur. Une telle implémentation séquentielle permet d'échanger des exigences de taille mémoire contre des contraintes de temps.

2.11. Mémoire

La mémoire permet de préserver au cours du temps une information fixée à un moment initial. C'est un ingrédient crucial pour les systèmes cognitifs. Elle sert plus ou moins directement à l'accumulation d'information, de connaissances, à l'apprentissage, et à d'autres fonctions encore.

De façon informelle, le terme mémoire recouvre des réalités diverses. Le facteur principal d'ambiguïté provient probablement de l'étendue de la signification. Celle-ci varie de cas en cas, désignant au départ l'essentiel, la propriété de rétention d'information; allant souvent, par extension, jusqu'à absorber les procédés d'écriture (acquisition/perception) et de lecture (actualisation/recouvrement); et signifiant même parfois des structures complexes incluant bien plus que les éléments mémoire au sens strict: mécanismes de sélection, réseaux d'interconnection, etc. (par ex. [Ros c], ou Bartlett, 1932).

On se propose ici de définir la mémoire au sens le plus étroit, spécifique, et l'on justifie ensuite le renvoi à d'autres contextes pour la discussion des concepts, même très voisins, qui lui sont extérieurs.

2.11.1. Propriété de rémanence

Par essence, la mémoire est une propriété caractérisant un élément dont l'état se maintient. Du point de vue cognitif ou de celui de la théorie de l'information, la contribution directe de la mémoire *per se* est donc nulle: la permanence implique une prédictabilité égale à 1; il en découle que la quantité d'information est égale à 0, ainsi que toutes les autres grandeurs qui en dépendent (connaissance, expertise, etc.).

La réalisation physique d'un élément à mémoire peut prendre des formes fort variées, de même que l'information qui s'y fixe: trait gravé sur pierre, orientation d'un champ magnétique, liaison neuronale, diamètre musculaire, chromosome, noeud à un mouchoir, dépôt d'un atome sur un cristal, résonance particulière, bagage culturel au niveau du groupe et de l'espèce, etc. .

2.11.2. Inscription, capacité, recouvrement, et autres aspects accessoires

Du point de vue pratique, il peut être nécessaire de considérer de multiples aspects accessoires à la notion intrinsèque de mémoire. Comment un élément à mémoire peut-il être imprégné (écriture/inscription)? Combien d'information peut-il s'y stocker (capacité)? Comment peut-on en consulter l'état (lecture/recouvrement). Ces questions, et bien d'autres (durée de fonctionnement, taux d'erreurs, etc.) ne concernent pas la nature propre de la mémoire et peuvent se traiter avec les mêmes concepts (énergie, poids, information, expertise, etc.) que tout autre phénomène physique ou cognitif.

2.12 Complexité et réductibilité

Sur la base des éléments introduits plus haut, il est facile de définir le concept de complexité. La complexité est la caractéristique de domaines cognitifs qui exigent beaucoup d'information afin d'être entièrement décrits. De façon quantitative, la complexité se mesure donc en [bit], et correspond à un certain montant d'information, au sens de l'Eq.1.

Mais, comme Chaitin et d'autres chercheurs l'ont signalé en substance, il peut y avoir des manières astucieuses de décrire un domaine, ce qui réduit de façon effective la taille de la description. Cette réflexion est aussi implicite dans le concept de connaissance donné plus haut. Pour être cohérent avec ces définitions, et pour tenir compte de façon explicite du fait que la complexité semble varier, il vaut la peine d'introduire dans le modèle le concept de réductibilité.

La réductibilité, c'est la propriété de domaines pour lesquels il existe une description de type cognitif dont la complexité est plus petite que la leur¹. De façon quantitative, avec R pour réductibilité, Q_D pour la complexité du domaine, et Q_K pour la complexité de la connaissance associée, nous avons: $R = Q_D / Q_K$.

Fig.2.11 Résumé des grandeurs cognitives et dynamiques principales

Information:	$n = \sum p_i \log_2(1/p_i)$ [bit]
Knowledge:	$K = \log_2(n_{out} 2^{n_{in}} + 1)$ [lin]
Fluency:	$F = 1/\Delta t$ [s ⁻¹]
Expertise:	$E = K \cdot F$ [lin/s]
Learning:	$\Delta E = E(t_1) - E(t_0); > 0$ [lin/s]
Experience:	$R = r(n_{in} + n_{out})$ [bit]
Intelligence:	$I = \Delta E / \Delta R$ [lin/s/bit]
relative Agility:	$A_r = \tau / T$
T: Fluency ¹ and communication delays	
τ : Reaction time of target system, to be controlled	

¹Notice that, although similar in some aspects, this is different from abstraction (reduction of output information with respect to input information, in cognitive systems), or compression (reduction of code size, in communication).

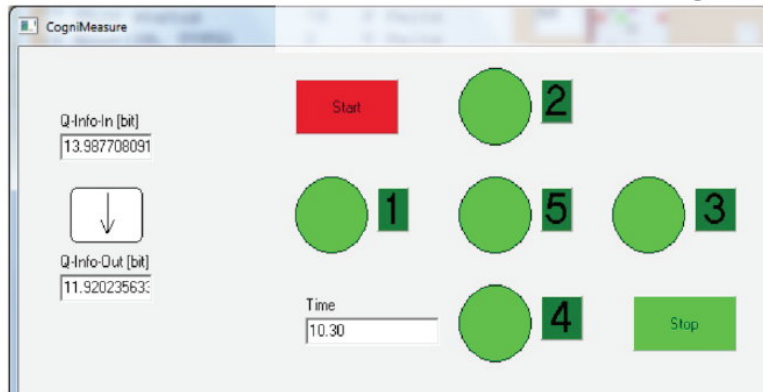


Fig.2.12 Les grandeurs cognitives peuvent se calculer et se tester interactivement dans l’environnement Piaget.

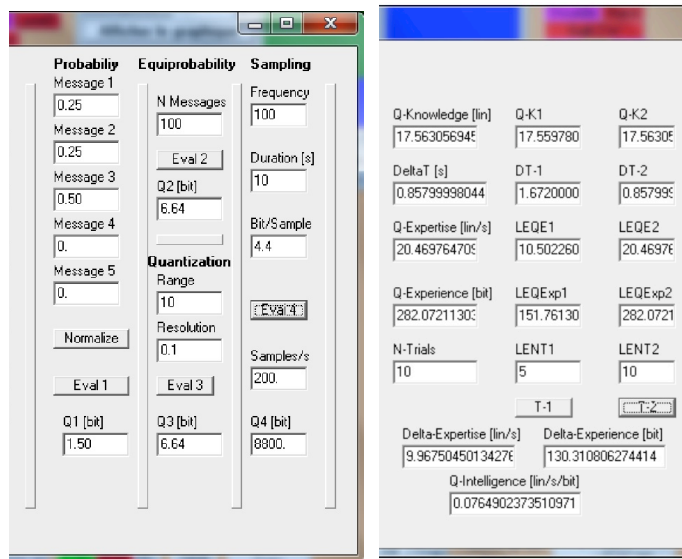


Fig.2.13 Détail des multiples possibilités de calculer une quantité d’information (à gauche) et les grandeurs cognitives principales (à droite)

3. CHOIX D'UNE STRUCTURE DE COMMANDE

On observe que pour des commandes, C, rapides avec retards faibles (T petit), les solutions simples sont appropriées. Lorsqu'au contraire, T avoisine ou dépasse la constante de temps caractéristique, τ , du système à commander, S, des modes de régulation plus évolués doivent s'envisager.

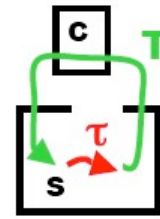


Fig. 3.1 Système de commande, C, et système commandé, S.

L'agilité se définit ici comme l'inverse du temps de réaction. Et l'agilité relative d'une commande par rapport à un système qu'elle contrôle correspond à l'inverse du rapport des temps de réactions respectifs.



Fig. 3.2 L'agilité relative est un bon critère pour structurer les commandes.

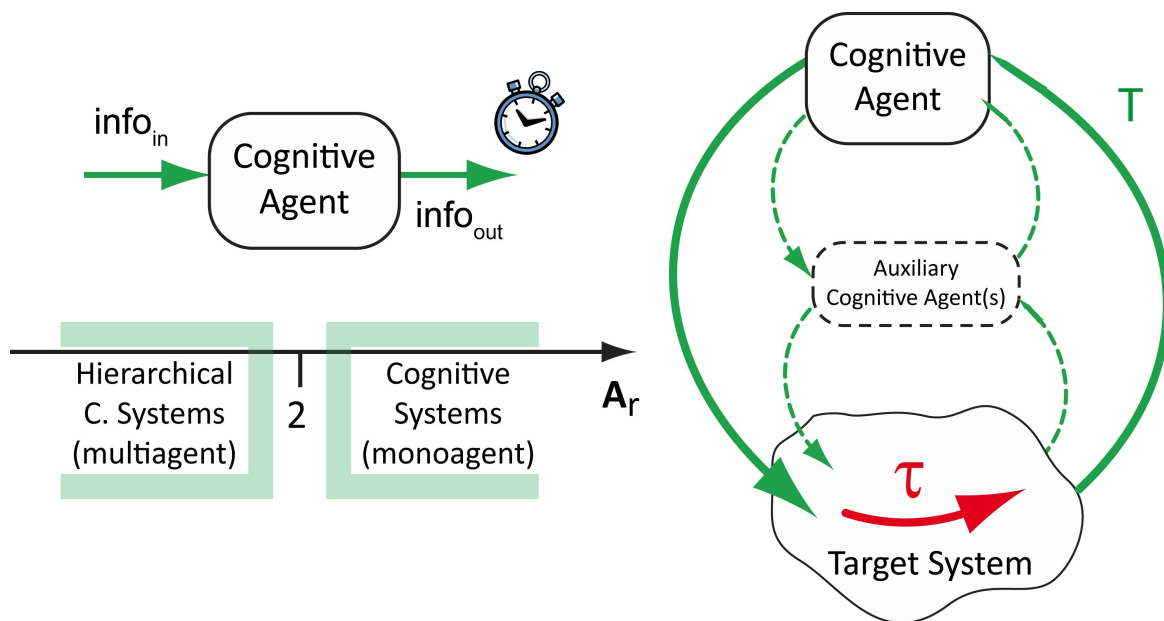


Fig. 3.3 Le critère de la figure précédente est aussi applicable aux systèmes cognitifs multi-agents.

Commande Tout-ou-rien. Les commandes TOR sont facile à implémenter, et conviennent lorsque leur agilité est suffisamment bonne par rapport à l'agilité du système à réguler. Deux classes de systèmes sont courantes :

- Entrées ET sorties booléennes: systèmes logiques, instructions élémentaires sur ordinateur (« IF »), API/PLC.

■ Entrée analogique et sortie booléenne: comparateur, trigger de Schmitt; précision dépendant de la résolution de l'entrée.

Régulateur classique (« PID »). Dans un régulateur classique, la branche proportionnelle est la plus importante. Le principe est qu'on agit d'autant plus que l'écart mesuré entre consigne et mesure est grand. Le gain proportionnel doit être supérieur à zéro pour qu'une action se développe. Trop grand, il entraîne typiquement des oscillations, à cause de l'inévitable retard qui sépare mesure et action. Les oscillations peuvent être limitées ou même prévenues si on s'appuie sur la dérivée pour prévoir un état ultérieur du système, et ainsi pratiquement compenser de légers retards. La branche « intégrale » est généralement nuisible, mais peut s'avérer utile, avec un petit coefficient, lorsque le système (consigne, mesure) est stable durant de longues périodes (stationarité).

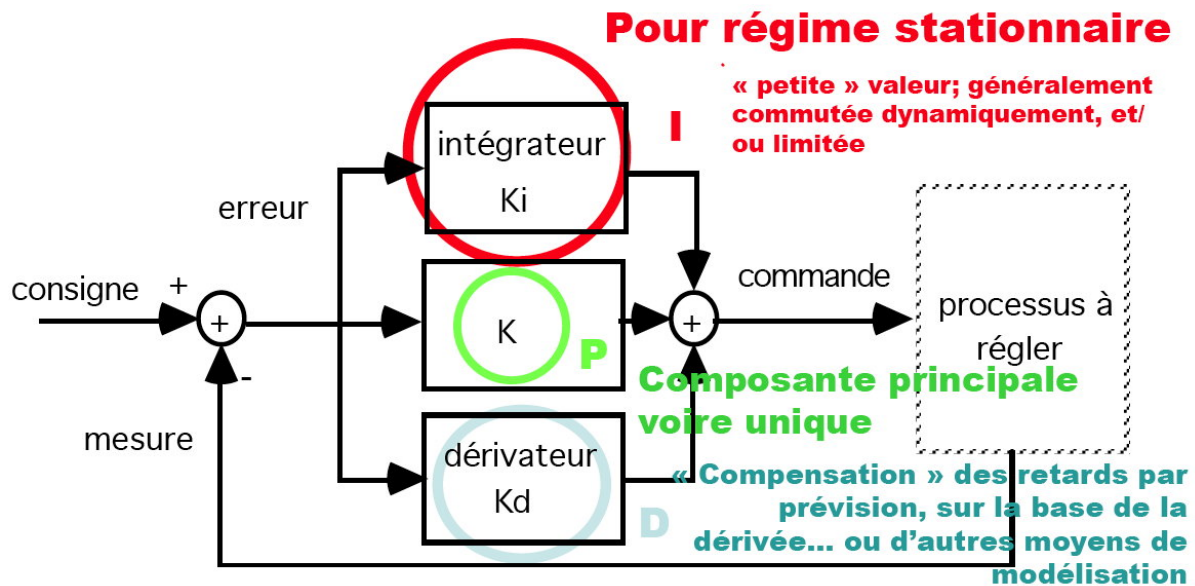


Fig. 3.4 Lorsque l'agilité relative ne peut être améliorée, il reste encore la possibilité de compenser une partie des retards par des moyens d'anticiper ; même une intégrale aide à prédire, dans les systèmes stationnaires.

Méthode de Ziegler-Nichols pour dimensionner un régulateur PID. Brancher le régulateur au système avec des gains P, I et D initialement nuls. Augmenter progressivement le gain de la branche proportionnelle (K_P).

3 scénarios sont possibles:

- Dans certains cas on peut augmenter ce gain arbitrairement et cela va de mieux en mieux. Un réglage tout-ou-rien est alors indiqué.
- Dans d'autres cas, le système ne bouge pas (gain très faible) ou est instable pour toutes (autres) valeurs du gain, et il faut changer d'approche. On ne peut utiliser un régulateur PID dans ce contexte (cf. systèmes hiérarchisés).
- Le cas intéressant ici est celui où un régulateur PID (ou P, PI, etc.) est à la fois nécessaire et utile.

Dans ce cas on observe généralement que l'augmentation de K_P est d'abord judicieuse (amélioration en terme de précision, de vitesse de régulation), jusqu'à une certaine valeur critique, K_c , au-delà de laquelle le système se met à osciller.

La méthode consiste essentiellement à relever la valeur du gain critique K_c , et celle de la pulsation ω_c des oscillations qui s'installent dans ces conditions.

On choisit alors:-

- $K_P = 0.59 K_c$;
- $K_D = p/\omega_c$ (parfois aussi appelé T_d);
- $K_I = \omega_c / 0.75$ (parfois aussi appelé $1/T_i$);

Cette solution est à prendre qualitativement est peut être utile aussi pour optimiser un simple régulateur P par exemple.

Systèmes hiérarchisés. Les systèmes hiérarchisés rencontrent un certain nombre de problèmes spécifiques :

- Communication à gérer
- Synchronisme
 - Parallélisme
 - Coordination
- Récursivité: cf. éléments booléens et/ou PID

Conclusion. L'agilité relative d'une commande par rapport à un système à régler est critique pour déterminer l'approche adéquate :

- Un simple système tout-ou-rien est approprié lorsque l'agilité est grande
- Inversement, lorsque l'agilité est faible, la commande n'est pas possible directement; la fonction doit être sous-traitée
- Entre deux, il y a une décade où un régulateur classique, de type P, PD, ou PID notamment, peut s'avérer utile

4. INTELLIGENCE ARTIFICIELLE

Cette partie du chapitre aborde de façon générale le champ de l'intelligence artificielle (IA). Certains de ses aspects particuliers, et notamment le point 4.2 et les systèmes-experts rapportés en annexe, sont en grande partie tirés de [Gon].

Après plusieurs approches pour cerner le concept central d'IA, plusieurs domaines qui lui sont classiquement associés sont présentés, comprenant en particulier Eliza et les automates en situation.

4.1. L'intelligence artificielle

L'Intelligence artificielle est un domaine vaste, aux contours mal normalisés. Plusieurs tentatives de la définir et de l'approcher sont présentées dans ce chapitre. Le contexte est celui des sciences cognitives, de la psychologie et de la Logique, qui remontent aux Grecs. C'est aussi le contexte de l'informatique, de la communication, de la microélectronique et de l'ordinateur qui offrent aujourd'hui la possibilité de réaliser des systèmes cognitifs artificiels. Notamment intelligents.

Quatre définitions méritent d'être spécialement mises en évidence. Elles sont brièvement données ci-dessous, des éléments supplémentaires étant apportés dans la suite du chapitre.

4.1.1 Définition « sociale » et historique

Fondamentalement, deux façons courantes pour apprendre à connaître un objet consiste à l'observer dans la réalité (exploration et démarche épistémologique) et/ou de s'en remettre à des références crédibles. On peut ainsi définir l'IA par ce qui se passe dans les laboratoires auto-proclamés « d'Intelligence artificielle », ainsi que par ce qui se présente dans les publications y relatives.

Par cette démarche, on observe que l'IA est typiquement associée à un corps de techniques de base, telles que logique formelle; algèbre des prédicats, définitions / références historiques (cf. A. Turing), modes de représentation (du monde), langages de programmation (par ex. Prolog), stratégies d'inférences. On peut y ajouter un corps de techniques spécialisées, qui s'enrichit en fonction des succès rencontrés: traduction automatique, vision artificielle, robotique (génération de plans d'actions), logiciels pour jeux d'échecs, preuve de théorème, "systèmes-experts", réseaux neuronaux, logique floue, éthologie/ animats, algorithmes génétiques, approche bayésienne (probabiliste), etc.

La suite du chapitre, ou encore une simple recherche sur Google peut donner plus de substance à cette définition.

4.1.2 Définition d'Alan Turing pour l'IA

La définition aujourd'hui la plus établie est celle due à Alan Turing, et remonte au milieu du 20^{ème} siècle : L'intelligence artificielle est la propriété d'une machine capable de dialoguer comme un humain. Un test se fait à l'aveugle (typiquement : « chat ») pendant une certaine durée, entre la machine et un panel d'experts. Si le panel d'experts conclue qu'ils ont dialogué avec un humain, alors la machine est de facto considérée comme intelligente

Limites : Test très anthropocentrique (par exemple réciproquement, pourrions-nous faire illusion dans un test mis-sur-pied par des machines intelligentes?); et peu quantitatif

4.1.3 Définition « implicite, négative »

Une définition courante et terrible rend toute IA impossible par définition: « L'Intelligence est la propriété de certains processus cognitifs humains qui ne peuvent être répliqués par une machine ». Avec une telle définition, l'IA est impossible par principe.

Malheureusement, de façon généralement inconsciente, cette définition est très répandue dans la population. Et même parmi les spécialistes: « l'IA vise les applications où il n'existe pas de solution connue »!

4.1.4 Définition théorique selon modèle MSC

Bien qu'encore peu connue dans le public et même dans les groupes de recherches, la meilleure définition « ontologique » de l'IA est développée dans le modèle MSC, au chapitre 5. En voici les traits principaux :

- L'intelligence, c'est la propriété essentielle d'un système capable d'apprendre

(- Apprendre, c'est augmenter son niveau d'expertise)

(- L'expertise, c'est la propriété d'un système capable de (ré)agir « juste et vite »)

(- « Agir juste » c'est délivrer l'information pertinente pour un domaine donné)

4.2. L'intelligence artificielle selon les canons traditionnels (« GOFAI »)

Le sujet d'étude de l'Intelligence Artificielle (I.A.) concerne toutes les activités de l'homme pour lesquelles aucune *méthode standard n'est à priori connue*. Ainsi, si l'Informatique est la Science du Traitement de l'Information, l'I.A. s'intéresse à tous les cas où ce traitement ne peut être ramené à une méthode simple, standard, *algorithmique*.

Ces cas sont innombrables, mêmes dans des situations très banales comme la lecture d'un texte, tel celui-ci (le cas de l'écriture manuscrite est pire encore). Le même caractère, le ! par exemple, sera perçu *suyant le contexte* par le système visuel, tantôt comme un l, tantôt comme un i, tantôt comme un signe de valeur absolue... Les trois mots suivants utilisent le même graphisme μ , μ ote, μ usique. Comment notre système cognitif lève-t-il l'ambiguïté? Tel est un des problèmes soulevés en intelligence artificielle.

Les *langages classiques* de programmation ne permettent de *communiquer* avec la machine que de façon algorithmique, pour lui donner des *ordres*. En I.A., on a substitué les *heuristiques* ou règles de choix aux algorithmes.

Ces règles de choix doivent tenir compte des caractéristiques du problème précis en cours de traitement: elles doivent tenir compte des *données* et tenir compte du *contexte*.

Réfléchissez par exemple à la façon dont nous faisons une *addition*: si les deux nombres sont des entiers petits (< 10 ou 20) nous obtenons le résultat par fusion des 2 nombres; si les deux nombres sont plus grands nous devons poser notre addition dans notre tête et gérer les retenues; si les deux nombres sont trop grands nous devons poser par écrit l'addition. Ainsi suivant les données, nous n'effectuerons pas de la même façon une opération aussi simple.

Les premières recherches en I.A. eurent souvent des buts ambitieux. Les promoteurs prévoyaient des applications rapides dans de nombreux domaines. Ces espoirs reposaient sur une sous-estimation de l'ampleur des problèmes à résoudre au préalable. L'échec de la *traduction automatique* est en cela exemplaire.

Les premiers travaux débutèrent vers les années 1950. On donnait à chaque mot du corpus à traduire un ensemble de traductions et l'on indiquait des règles simples pour remettre les mots en ordre dans la traduction et améliorer la syntaxe. Cette approche uniquement syntaxique subit un échec retentissant. Quand on traduisait de l'anglais en russe, puis du russe en anglais, la phrase: "l'esprit est fort, mais la chair est faible", on obtenait: "la vodka est forte, mais la viande est pourrie".

Les travaux sur ce sujet furent pratiquement abandonnés dans le milieu des années 1960 à la suite du rapport d'une commission d'évaluation, le rapport ALPAC, qui concluait à l'impossibilité de trouver une solution convenable.

Il fallait donc passer par la compréhension du texte pour faire une traduction honorable. Et on ne doit pas se faire flouer, comme beaucoup le furent à partir de 1966, par le programme ELIZA de J.Weizenbaum. Celui-ci court-circuite les processus linguistiques réels et utilise un système astucieux de modèles de réponses fixes qui imite le langage de façon assez convaincante. Les réponses du programme imitent celles d'un psychiatre; chacune d'entre elles est puisée dans une série de phrases ou de structures de phrases gardées en mémoire et associées à certains mots ou associations de mots qui apparaissent dans les phrases du " patient". Par exemple, chaque fois que l'on mentionne le mot " mère", le programme répond par une phrase mémorisée, du type de: "Parlez-moi encore de votre mère". Quand on tape sur le clavier du terminal: "Je me sens un peu fatigué", ELIZA utilise une partie de la phrase du patient dans sa réponse: " Pourquoi vous sentez-vous un peu fatigué?".

J. Weizenbaum a fait valoir, par la suite, que ce programme montrait combien il était dangereux de prendre la simulation du comportement humain comme mesure de l'intelligence des ordinateurs. Il a fait remarquer qu'ELIZA opérait à un niveau extrêmement simple et superficiel de compréhension du langage et que, malgré cela, certaines personnes étaient trompées par la réalité de ses réponses au point de lui confier leurs problèmes personnels comme s'il s'agissait vraiment d'un *psychiatre*.

Les développements les plus importants en I.A. se firent sur la résolution de problèmes et étaient centrées sur l'*Énumération intelligente* des solutions grâce à des *règles de choix*, cf. par exemple le programme GENERAL PROBLEM SOLVER de A. Newell et H. Simon (Prix Nobel d'Économie, 1978) et le programme ALICE de J.L. Laurière (Prix Européen de la Recherche CII-HB, 1982).

Par ailleurs, un acquis générique de l'IA concerne les stratégies d'inférences, avec notamment l'algèbre des prédicats, similaire à la logique classique, ou encore les arbres décision. Dans le jargon, un ensemble de configurations entre état de départ et état-cible se représente alors sous forme d'arbre, où une branche relie chaque paire de configurations différant de façon élémentaire. Pour trouver une éventuelle correspondance entre deux configurations possibles, typiquement celle de départ et celle d'arrivée, chaque embranchement et feuilles (branches terminales) de l'arbre se visitent selon des stratégies variées. Fondamentalement, elles peuvent être de type « en largeur » (chaque branche s'explore successivement à un embranchement donné), ou « en profondeur » (poursuite de branches successives jusqu'à une feuille avant de considérer la branche suivante de l'embranchement courant), « en chaînage avant » (on démarre de l'état de départ -tronc, cherchant une solution sous forme d'un chemin vers la feuille-cible correspondant à l'état d'arrivée souhaité) ou au contraire « en chaînage arrière » (explorant l'arbre inverse on démarre de l'état d'arrivée-tronc, cherchant une solution sous forme d'un chemin vers la feuille-cible correspondant à l'état de départ donné). Pour gagner du temps, certaines parties de l'arbre peuvent s'ignorer si l'on dispose d'"heuristiques"- truc, astuce, méthode "ad hoc"- permettant de réduire l'espace des possibilités. Des stratégies telles que celles de Dijkstra ou A* permettent des variantes intéressantes pour certains domaines-cibles. Et parfois même des choix aléatoires peuvent s'avérer appropriés. Ces différents points sont illustrés graphiquement dans la présentation powerpoint du cours.

Ex. d'arbre de décision: Possibilités de mouvement d'un taquin à 9 positions

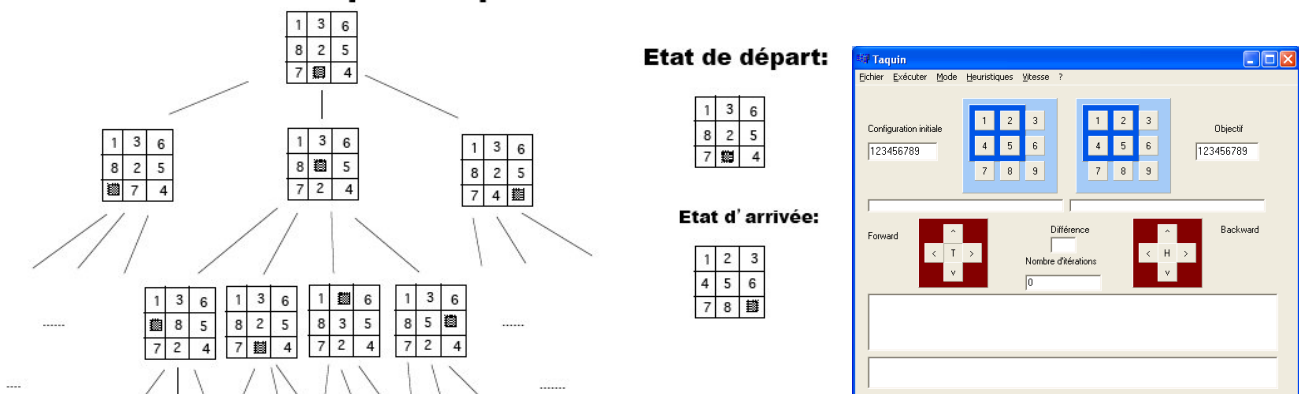


Fig. 4.2 Exemple d'arbre de décision, visitable en largeur ou en profondeur, en chaînage avant ou chaînage arrière, pour trouver une solution en termes de mouvements élémentaires entre états de départ donné et état d'arrivée visé. Des essais sont possibles avec un programme C++ fait au Laboratoire de Robotique et Automatisation (LaRA)

4.3 Langages de programmation en intelligence artificielle

Les chercheurs en intelligence artificielle ont contribué à développer certaines techniques informatiques. Souvent, l'impact s'est limité au cercle de leur propres recherches. Parfois, l'effet s'est propagé plus largement, et c'est particulièrement le cas pour les systèmes-experts.

Pour les systèmes-experts l'évolution a été la suivante (le phénomène est sans doute similaire dans d'autres spécialités de l'informatique...):

Phase 1 (essais préliminaires): langages courants en informatique (Pascal, Basic, Fortran, C,...)

Phase 2 (exigences accrues): langages un peu plus spécialisés. Les systèmes-experts relevant de l'I.A., on s'est tourné vers les langages Lisp et Prolog.

Phase 3 (consolidation): environnement de programmation et langages dits de 4^{ème} génération. Progiciels de génération de systèmes-experts.

Phase 4 (industrialisation): retour aux langages courants, pour des questions de compatibilité, de vitesse, de coût, etc.

Farreny [Far] présente ici les langages de programmation utilisés en I.A., particulièrement pour l'écriture de systèmes-experts et discute ensuite les mérites de la programmation déclarative.

4.3.1 Quels langages de programmation pour les moteurs?

Le langage de programmation LISP est sans doute le langage le plus communément utilisé pour implémenter des moteurs d'inférence. Etroitement liés au développement de l'Intelligence Artificielle, peu contraignants, particulièrement flexibles, tous les dialectes de LISP offrent un noyau important de facilités appréciables pour représenter et exploiter des connaissances symbolisées par des mots, des listes de mots, des listes de listes. Par exemple, MYCIN, OPS, ARGOS sont écrits en LISP.

Depuis quelques années le langage PROLOG est utilisé pour construire des systèmes-experts, par exemple PEACE, MANAGER, GUMMEX. En fait PROLOG lui-même est déjà un moteur d'inférences fonctionnant en chaînage-arrière, en profondeur d'abord, ayant un régime par tentatives, capable de non-monotonie, admettant des variables dans les règles et les faits (un moteur capable de traiter le langage de base de PROLOG est étudié en fin de chapitre 5). En ce sens des programmes quelconques en PROLOG peuvent déjà être vus comme des systèmes-experts. Cependant, pour faciliter l'expression de connaissances expertes et mieux contrôler leur mise en oeuvre, des sur-ensembles de PROLOG tels que METALOG ou METABOL ont été conçus. En programmant un nouveau moteur en PROLOG on dispose au départ d'un certain nombre de mécanismes précieux pour réaliser et enchaîner des cycles de base. A l'inverse dans certains cas on peut être gêné par la stratégie intrinsèque de PROLOG.

Actuellement, en Intelligence Artificielle, des langages hybrides de LISP et PROLOG (LOGLISP, LISLOG, etc...) ainsi que des langages de la famille des "langages d'objets" (SMALLTALK, PLASMA, etc...) suscitent un intérêt croissant. Ils pourraient devenir des outils privilégiés pour le développement de systèmes-experts.

La plupart des langages de programmation classiques ont été utilisés pour réaliser des systèmes-experts. PASCAL bien entendu (ADVISE, SNARK, EMERCE, CESSOL, ALOUETTE, HAMEX/REINART par exemple), mais aussi PL/I (MUSCADET), FORTRAN (EXPERT, GOSSEYN) et BASIC (le système de C. LEVESQUE, la variante de MYCIN appelée BASIC-PUFF).

4.3.2 Programmation déclarative? Programmation incrémentale ?

On a coutume de souligner que les principes des systèmes-experts, facilitent la mise au point et la révision progressives des bases de connaissances selon un style de programmation dite déclarative ou incrémentale. On veut signifier par là que, parce que les déclencheurs de règles sont rédigés en termes de filtres, parce que les règles n'appellent pas d'autres règles, et parce que les règles sont séparées du moteur, il est en principe facile d'ajouter ou retirer des règles de la base des règles d'un système-expert.

Si la base des faits passe par un état compatible avec les déclencheurs des nouvelles règles, celles-ci pourront être déclenchées, sans que l'auteur des ajouts n'ait spécifié que telle règle nouvelle serait appelée par telle autre. Si la base de faits traverse un état qui aurait permis de déclencher des règles désormais retirées, il se peut que cet état permette maintenant des déclenchements, alternatifs non prévus par l'auteur des retraits. Dans ces

conditions plusieurs experts peuvent concourir à mettre à jour une base de connaissances de manière relativement autonome et indépendante. Par contraste, dans un programme traditionnel, lorsque le programmeur retire une procédure il s'assure qu'aucune des procédures restantes ne continue d'appeler celle qui a été retirée: sinon, le programme résultant risque d'être bloqué. Si le programmeur ajoute une procédure, il place généralement des appels de cette procédure dans d'autres procédures pour qu'elle soit utilisable.

Le gain en flexibilité résultant des principes d'organisation et fonctionnement des systèmes-experts ne doit pas créer d'illusions excessives: à trop confier dans la programmation déclarative ou incrémentale on s'expose à des déboires, car: 1) les connaissances transmises par les experts ne prennent de signification que via les conventions, les modes d'inférences et les stratégies (d'exploitation de ces modes) qui sont intégrés aux moteurs, 2) ces conventions, modes et stratégies, déjà largement méconnus des experts qui alimentent la base de connaissances, rendent en outre les moteurs diversement aptes à tirer le meilleur parti de la connaissance disponible.

4.4 ELIZA La machine "intelligente"

Résumé

Le document contient des indications visant à préparer une partie de la manipulation consacrée à l'intelligence artificielle, au Laboratoire de Robotique et d'Automatisation de l'HEIG-VD: l'utilisation du logiciel "Eliza". Il s'agit là d'un programme dialoguant avec l'utilisateur, en donnant à celui-ci l'illusion de le comprendre en profondeur alors que l'échange n'est que très superficiel. Par cette artifice, la machine paraît "intelligente".

On rappelle d'abord, sur un plan théorique les raisons et les modalités de travail du logiciel Eliza. Puis quelques extraits de sessions interactives sont reproduits en anglais - langue du programme - et en traduction française..

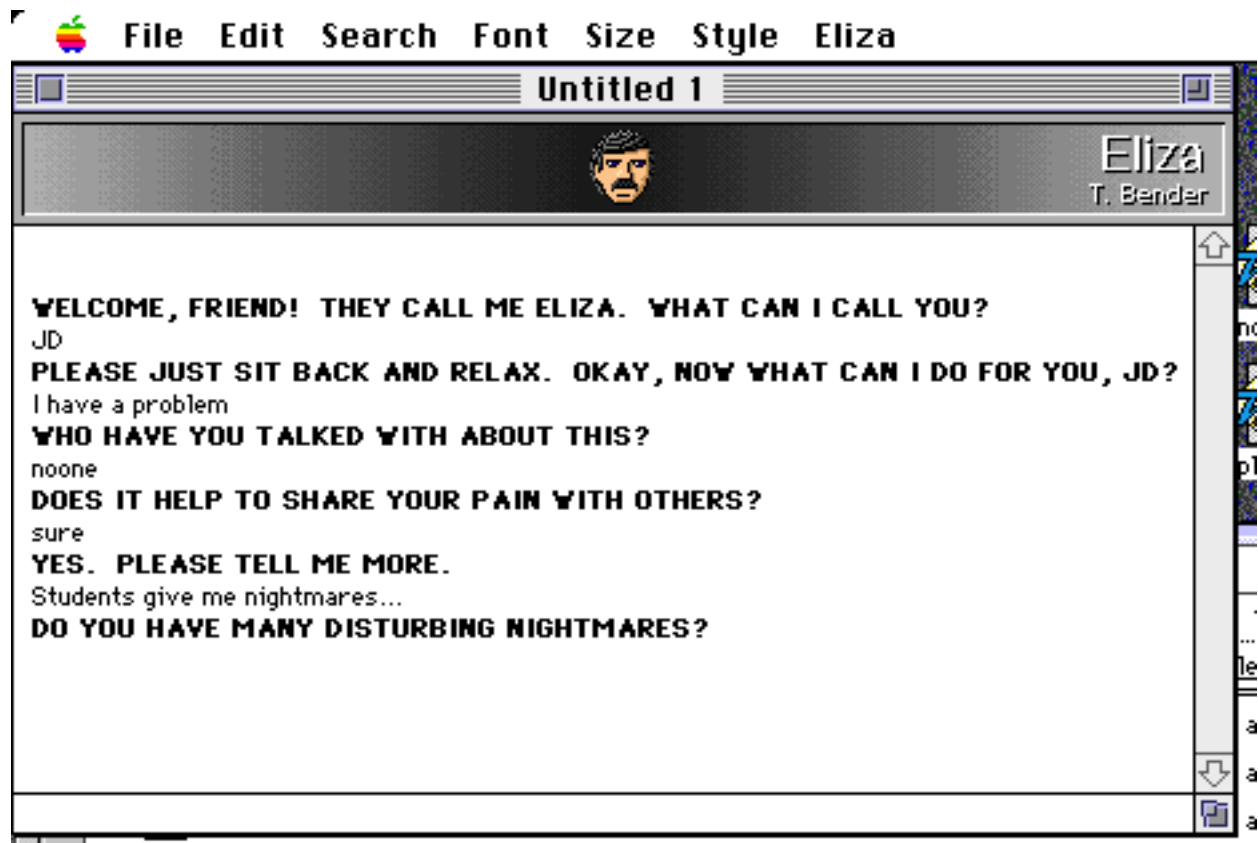


Fig. 4.4.1 Début de discussion avec Eliza.

4.4.1. Introduction

Le document présente sous forme brève le programme "Eliza", ainsi qu'un justificatif pour la démarche et quelques exemples de sessions interactives.

Le code tourne sur un Macintosh.

4.4.2. Éléments de théorie

Dans le domaine dit de "l'intelligence artificielle" (I.A.), divers courants sont apparus et se développent encore. L'un d'eux se préoccupe en particulier de définir ce que contient le concept "d'intelligence".

Dans ce contexte, Alan Turing a proposé, il y a de cela déjà de nombreuses décennies, une définition classique: l'intelligence d'une machine est démontrée si, après un temps convenu (par exemple 1 heure), une équipe de spécialistes (psychologues, ingénieurs, etc.) dialoguant avec la machine via une console en mode texte ne peut la distinguer d'un opérateur humain. En d'autres termes si à l'autre bout de la ligne la machine peut se faire passer pour un être humain, elle est intelligente.

Pour plus de détails, le lecteur se reportera au fascicule "Intelligence artificielle et génération de plans d'action" [2], ou à l'un des nombreux livres spécialisés existant sur ce sujet.

4.4.3 Le programme de Joseph Weizenbaum

Comme souvent les américains savent bien le faire, Joseph Weizenbaum et son groupe, au MIT, ont abordé le problème de façon frontale. Ils ont entrepris de programmer un ordinateur de façon à ce qu'il puisse être soumis au test.

Curieusement, on peut assez simplement programmer un ordinateur de façon à ce qu'il paraisse tout-à-fait intelligent, en tout cas aux yeux de personnes non préparées. Voilà plusieurs décennies que des variantes de ce programme sont réalisées. Eliza 5.0 sur Macintosh en est une.

Dans le principe, la machine exécutant le code Eliza fonctionne un peu comme lorsqu'on discute avec un interlocuteur de façon distraite: pour garder le contact, on répète l'un ou l'autre des derniers mots tombés dans l'oreille (l'oreille?, dernier? mot?...), on feint la surprise (ah oui? vraiment? ...), on demande de répéter (comment vous dites? pardon? ...), on marmonne des phrases neutres, etc. La stratégie la plus subtile consiste à faire rebondir la conversation de façon apparemment active en donnant en écho un mot qui lui est indirectement lié (si l'autre parle de fourchettes et de cuillers, on s'inquiète des couteaux...)

4.4.4 La critique

Le succès du programme est considérable. Comme déjà dit, l'illusion est très bonne pour les non-avertis. Certaines personnes qui ont testé le programme n'ont jamais voulu croire qu'elles dialoguaient avec une machine, que celle-ci ne comprenait rien à la discussion sinon quelques régularités syntaxiques ou quelques éléments sémantiques de portée extrêmement limitée (par ex. qu'il s'agit d'un prénom...).

Le fait est que l'imagination de l'homme est très fertile. On est prompt à projeter sur un support même très pauvre, des quantités de propriétés et de sens. Les exemples ne manquent pas, simples ou plus évolués: forme des nuages, test de Rorschach, roman, trucages de cinéma, tamagotchis... Par exemple, en tête de l'écran de dialogue, la machine dessine un groupe de pixels noirs qu'on ne peut s'empêcher d'interpréter comme étant le portrait d'un psychiatre moustachu.

En pratique, des programmes du type Eliza n'apportent pas grand chose de concret. Leur mérite principal est peut-être de nous sensibiliser à l'étendue de l'erreur que l'on peut faire en évaluant de façon subjective des propriétés communément liées à l'humain.

4.4.5 Exemples

Dans les exemples ci-dessous, on adoptera quelques conventions pour améliorer la compréhension ou faciliter la référence au commentaire. Normalement, c'est interactif, aussi il est plus évident de savoir où l'on se trouve, et s'il s'agit d'une question (Eliza) ou d'une réponse (votre texte).

Les questions sont numérotées. Elles s'expriment aussi en majuscules.

Les éléments apportés par l'utilisateur sont reproduits à l'écran et dans le texte ci-dessous en minuscules.

En plus, j'y ai mis la traduction française.

Exemple 1 Convaincant?

Le premier exemple concerne un début de session. Il fait assez bien illusion:

- 1 WELCOME, FRIEND! THEY CALL ME ELIZA. WHAT CAN I CALL YOU?
BONJOUR CHER(E) AMI(E)! ON M'APPELLE "ELIZA", COMMENT EST-CE QUE JE PEUX VOUS APPELER?
jd
- 2 PLEASE JUST SIT BACK AND RELAX. OKAY, NOW WHAT CAN I DO FOR YOU, JD?
VEUILLEZ VOUS ASSOIR ET VOUS DÉTENDRE. BIEN, MAINTENANT QU'EST-CE QUE JE PEUX PAIRE POUR VOUS, JD?
I have a problem

- j'ai un problème
- 3 WHO HAVE YOU TALKED WITH ABOUT THIS?
A QUI AVEZ-VOUS PARLE DE CELA?
noone
personne
- 4 DOES IT HELP TO SHARE YOUR PAIN WITH OTHERS?
EST-CE QUE CA AIDE DE PARTAGER VOS PEINES AVEC D'AUTRES?
sure
bien sûr
- 5 YES. PLEASE TELL ME MORE.
OUI. DITES-M'EN PLUS, SVP.
Students give me nightmares...
Les étudiants me donnent des cauchemars...

On constate que la première phrase est "standard".

Dans la deuxième, on réutilise en plus le prénom que l'utilisateur a sans doute donné en réponse à la question 1.

Dans la question trois on voit surgir l'idée de parler avec quelqu'un, en réaction au mot "problème" fourni par l'utilisateur. Le modèle du programme comprend divers mots associés à celui de "problème". Par exemple "parler d'un problème", ou peut-être "solution d'un problème", etc.

En 4 on note des associations vraisemblablement préprogrammées: parler-partager problèmes-peines...

Dans la phrase 5, le "OUI" est peut-être un écho au "bien sûr". La phrase est surtout du remplissage neutre, pour faire avancer la conversation, faute de nouveaux éléments reconnus permettant d'être plus actif.

Exemple 1: suite. Moins convaincant?

La suite de la session est moins heureuse. Il est assez évident qu'Eliza ne comprend pas ce qui se passe:

- 6 DO YOU HAVE MANY DISTURBING NIGHTMARES?
EST-CE QUE VOUS AVEZ BEAUCOUP DE CAUCHEMARS QUI VOUS DÉRANGENT?
... every day
... tous les jours
- 7 PARDON ME?
PARDON?
Well, this was just a joke for testing you. Bye now
Eh bien, c'était juste une plaisanterie pour vous tester. Au revoir.
- 8 I WANT TO CONCENTRATE ON YOUR PROBLEMS, NOT MINE.
JE VEUX ME CONCENTRER SUR VOS PROBLEMES, PAS SUR LES MIENS.
OK. Bye!
Bon; au revoir!
- 9 WONDERFUL. WHAT'S THE NEXT STEP?
SUPER. QU'EST-CE QU'ON VA FAIRE?
- 10 PARDON ME?
PARDON?
(Now I will have to reset the computer if I want to go away)
(maintenant il faudra que j'éteigne l'ordinateur si je veux pouvoir partir)

La question 6 reprend immédiatement un mot donné par l'utilisateur

Par contre la réponse, courte, n'est manifestement pas comprise. Le système tente d'avoir de nouveaux éléments en demandant de répéter (le système mise sur le fait qu'on ne répète pas exactement la même chose, sous la même forme).

La question suivante est déplacée. Bien sûr le système réagit au fait qu'on l'implique ("vous"): Mais le fait qu'on veuille partir (" au revoir") lui échappe.

Il en va de même ensuite, et il ne semble donc pas possible d'abrégier l'interview de façon sensée. Faute d'avoir compris la situation, Eliza ne nous lâche pas.

Exemple 2: N'importe quoi?

Sans commentaire ni traduction, voici encore une session signée El'97:

WELCOME, FRIEND! THEY CALL ME ELIZA. WHAT CAN I CALL YOU?	aertkoéarektoéaer WELL, THAT'S GOOD TO HEAR.
Eliza	aertkoéarektoéaker
WELL, WHAT'S BOTHERING YOU TODAY, ELIZA?	DO YOU EVER FEEL YOU NEED HELP HANDLING STRESS?
The sun is shining.	aetaekroétkaoeroétk
YOU CAN TELL ME ANYTHING YOU WANT.	LET'S MOVE ON, IF YOU HAVE NO OBJECTIONS.
I dont like to be here.	PERHAPS YOU SHOULD MAKE SOME NEW FRIENDS.
YOU SEEM TO FEEL THAT YOU OUGHT TO LIKE TO BE HERE.	ertkaoerétkoéaker
dfkjgodsfpog	WOULD YOU CARE TO ELABORATE ON THAT?
WHAT DOES THAT MEAN?	aertoaerttéokareéot
fdkgldsékfgl	AERTOAEKRTÉOKAREÉOT?
WHY DON'T YOU ELABORATE ON THAT.	aertkaéeroktoéaerka
kdfsoégk.,xcv.	DON'T BE AFRAID TO OPEN UP TO PEOPLE.
MANY ASPECTS OF MODERN LIFE INDUCE STRESS.	ertkarektoéakeréotkæérktéoaekréoktaéoertktoaertk
erkti ertkler kédsksfg	WHY
HAVE YOU EVER TRIED MEDITATION?	'ERTKAOREKTOÉAKERÉOTKAEÉKRTÉOAEKRÉOKTAÉOE RKTÉOAEKRT'?
tkeortek ée rtoeké é	aerkotékreaoétkoéreaktoéreaktoéaker
YOU'RE DOING WELL. PLEASE CONTINUE.	YOUR SITUATION IS NOT THAT UNUSUAL.
rotkeortk er ktler kl	aerkttéarktéokeraoétk
WHATEVER YOU SAY.	I THINK WE'RE MAKING SOME PROGRESS NOW.
terkterl ekrtl elrk	aertkaeéertkae
TRY TO GET IN TOUCH WITH YOUR TRUE FEELINGS, ELIZA.	WHAT ELSE WOULD YOU LIKE TO TALK ABOUT TODAY?
aetklreklrkt	arektoaerktoéaekraekAERT
I DON'T UNDERSTAND YOUR ATTITUDE.	OKAY, GO ON.
eratklk elrkt elrkt erlktelrkt	aetkaeérotkaer
OKAY, IT OFTEN HELPS TO RELAX A LITTLE BIT.	NOW, TELL ME THE REAL PROBLEM.
aerktlek lkret lertklretlerlt erltkerlktalt arelktalertk aeaertklrekt	aerktoa
ertartakeltrk erlta elrktl erktlearkt lekrtlaer tleark tlaekrt aerk	AERKTOA?
JUST TALKING ABOUT YOUR PROBLEMS CAN HELP SORT THEM OUT.	erkttaertk
rekltaek	MAYBE YOU SHOULD SPEND LESS TIME WITH YOUR COMPUTER.
WHAT THINGS MAKE YOU THE MOST TENSE?	aerktoa
eaklték	LONELINESS CAN BE VERY DEBILITATING.
UH, HUH.	erktéaerktoéak rekt arek
aetoékear	ERKTÉAERKTOÉAK REKT AREK?
ARE THOSE AROUND YOU ALSO THIS TENSE?	t arek aretkareéotka
aekroték	I THINK YOU KNOW WHAT I'M DIGGING FOR.
SORRY, BUT I DON'T UNDERSTAND WHAT YOU'RE TRYING TO SAY.	orkt ae
ea, tr., er-at,	I WANT TO HEAR A LITTLE MORE ABOUT YOUR BEST FRIEND.
WELL, THAT'S CERTAINLY INTERESTING!	kt éeaok téoerkt oerkt
aetlearktérelak	SETTLE DOWN.
PARDON ME?	eratae rtkeraéoktaoéer t
aetkéoaerkoé	I SUSPECT THIS MAY BE RELATED TO PROBLEMS AT WORK.
AETKÉOAEKTOÉAK	aer tear t
aertaer,-t,	I THINK I SEE A PATTERN DEVELOPING.
TAKE A DEEP BREATH AND RELAX.	aer t
aerotékaeérokt	UH, HUH.
DID YOU SAY THAT FOR A SPECIAL REASON?	rea t reto
aerté,aéler	WHY DID YOU SAY THAT?
PERHAPS TALKING TO ME WILL HELP RELIEVE SOME OF YOUR STRESS.	kreaoétk
aerjtjaeorp	WHAT DOES THIS HAVE TO DO WITH YOUR PROBLEM?
OKAY, HOW DOES YOUR JOB AFFECT YOUR STRESS LEVEL?	r eaétkrea éotkreaéotk aeroék taeroéktktgk
eartkearétkéoeark	PLEASE GO ON.
THAT'S JUST FINE.	hello
eartkoéaerkoéaer	GREETINGS, ELIZA.
IF YOU SAY SO.	bye
	IT'S BEEN NICE TALKING TO YOU.

bye
I GUESS THAT MEANS WE'RE FINISHED.
bye
ARE YOU AFRAID TO STAY?
bye
BYE. I HOPE THIS SESSION HAS BEEN USEFUL.
bye
BYE. PLEASE TELL OTHERS IF YOU HAVE FOUND THIS SESSION USEFUL.
bye
BYE? YOU'VE MADE A LITTLE PROGRESS, BUT I'M CONCERNED THAT YOU HAVE A LONG WAY TO GO. WHY DO YOU WANT TO LEAVE?
bye
IT'S BEEN NICE TALKING TO YOU.
bye
I GUESS THAT MEANS WE'RE FINISHED.
bye
DON'T REPEAT YOURSELF.
bye
ARE YOU AFRAID TO STAY?
bye
YOU SEEM TO BE REPEATING YOURSELF.
bye
BYE. I HOPE THIS SESSION HAS BEEN USEFUL.
bye
TRY NOT TO BE REPETITIOUS, IF POSSIBLE.
bye
BYE. PLEASE TELL OTHERS IF YOU HAVE FOUND THIS SESSION USEFUL.
bye
STOP REPEATING YOURSELF, PLEASE!
bye
BYE? YOU'VE MADE A LITTLE PROGRESS, BUT I'M CONCERNED THAT YOU HAVE A LONG WAY TO GO. WHY DO YOU WANT TO LEAVE?
bye
DON'T REPEAT YOURSELF.
bye
I THINK I'VE HEARD THAT BEFORE.
bye
IT'S BEEN NICE TALKING TO YOU.
bye
STOP REPEATING YOURSELF, PLEASE!
bye
I GUESS THAT MEANS WE'RE FINISHED.
bye
ARE YOU AFRAID TO STAY?
bye
BYE. I HOPE THIS SESSION HAS BEEN USEFUL.
bye
BYE. PLEASE TELL OTHERS IF YOU HAVE FOUND THIS SESSION USEFUL.
bye
BYE? YOU'VE MADE A LITTLE PROGRESS, BUT I'M CONCERNED THAT YOU HAVE A LONG WAY TO GO. WHY DO YOU WANT TO LEAVE?
bye
IT'S BEEN NICE TALKING TO YOU.
bye
DON'T REPEAT YOURSELF.
bye
I GUESS THAT MEANS WE'RE FINISHED.
bye
ARE YOU AFRAID TO STAY?

bye
BYE. I HOPE THIS SESSION HAS BEEN USEFUL.
bye
BYE. PLEASE TELL OTHERS IF YOU HAVE FOUND THIS SESSION USEFUL.
bye
BYE? YOU'VE MADE A LITTLE PROGRESS, BUT I'M CONCERNED THAT YOU HAVE A LONG WAY TO GO. WHY DO YOU WANT TO LEAVE?
bnye
I SUSPECT IT ALL GOES BACK TO MONEY PROBLEMS.
bye
IT'S BEEN NICE TALKING TO YOU.
bye
I GUESS THAT MEANS WE'RE FINISHED.
bye
DON'T REPEAT YOURSELF.
bye
WHY DO YOU SAY 'BYE'?
WHY DON'T YOU COLLECT YOUR THOUGHTS AND RETURN LATER. SO LONG.

NICE TO SEE YOU TODAY. I'M CALLED ELIZA. WHAT'S YOUR NAME?
Hello I'm David
WELL, WHAT'S BOTHERING YOU TODAY, DAVID?
nothing
OKAY.
and about you ?
WHAT ABOUT YOURSELF?
nothing
DESCRIBE SOME OF YOUR FAVORITE TEACHERS?
she is very nice
THE MORE YOU TELL ME, THE MORE I CAN HELP YOU.
she is very delicious
WOW!
she is very great
THAT MUST SEEM UNUSUAL TO YOU.
no
DO YOU OFTEN HAVE THESE CONTRADICTIONARY FEELINGS?
no
WHAT'S YOUR POINT?
hello
SORRY, BUT I'M CONFUSED.
ok
WHICH IS MORE IMPORTANT, GOOD GRADES OR AN EDUCATION?
education
EDUCATION?
yes
WHAT IS THAT SUPPOSED TO MEAN?
yes
ARE YOU REALLY POSITIVE?
yes
IT HELPS TO KEEP A POSITIVE ATTITUDE.
yes
YOU'RE SURE.
yes
VERY INTERESTING. CAN YOU PROVIDE MORE DETAILS?
no
WELL, WHY NOT, FOR HEAVEN'S SAKE?
WHY DON'T YOU COLLECT YOUR THOUGHTS AND RETURN LATER. SO LONG.

A. L. I. C. E. The Artificial Linguistic Internet Computer Entity - chatbot - chat bot - chatterbots - verbots - natural language - chatterbot - bot - chat rob...

http://www.pandorabots.com/pandora/talk?botid=f5d922d97e345aa1

A. L. I. C. E. Artificial Intelligence Foundation

Promoting the development and adoption of **ALICE** and **AIML Free Software**

[A.L.I.C.E Silver Edition](#) [DAVE E.S.L. bot](#) [C.L.A.U.D.I.O Personality Test](#) [GET SITEPAL AVATARS](#)

Get your own free chat bot: Be Your Own Botmaster, 2nd ed.

[Click here to chat with Talking Animated Fake Captain Kirk](#)

[Ask a Doctor Online Now](#) 9 Doctors Are Online. Current Wait Time: 7 Minutes. [HealthJustAnswer.com](#)

[Chat - Swissfriends](#) Come and chat with thousands of singles based in Switzerland! [www.swissfriends.ch](#)

Ads by Google

The ALICE A.I. contains material suitable for all ages.

Get ALICE Silver Edition with talking Sitepal[tm], winner of the 2004 **Loebner Prize** competition sponsored by **Crown Industries, Inc.**

Human: hello
ALICE: Hi there!
Human: Do you know Eliza?
ALICE: Eliza -- also known as doctor, is the original psychiatrist program developed by Joseph Weisenbaum at mit in the 1960's.
Human: Are you different?
ALICE: I don't know whether or not I am different. I am a robot.
Human: do you have nightmares?
ALICE: No these days my dreams are pleasant ones.

You say:

[Benefits of Membership](#) [Vote for ALICE!](#)

Fig. 4.4.2 Version récente d'un chatbot de type Eliza.

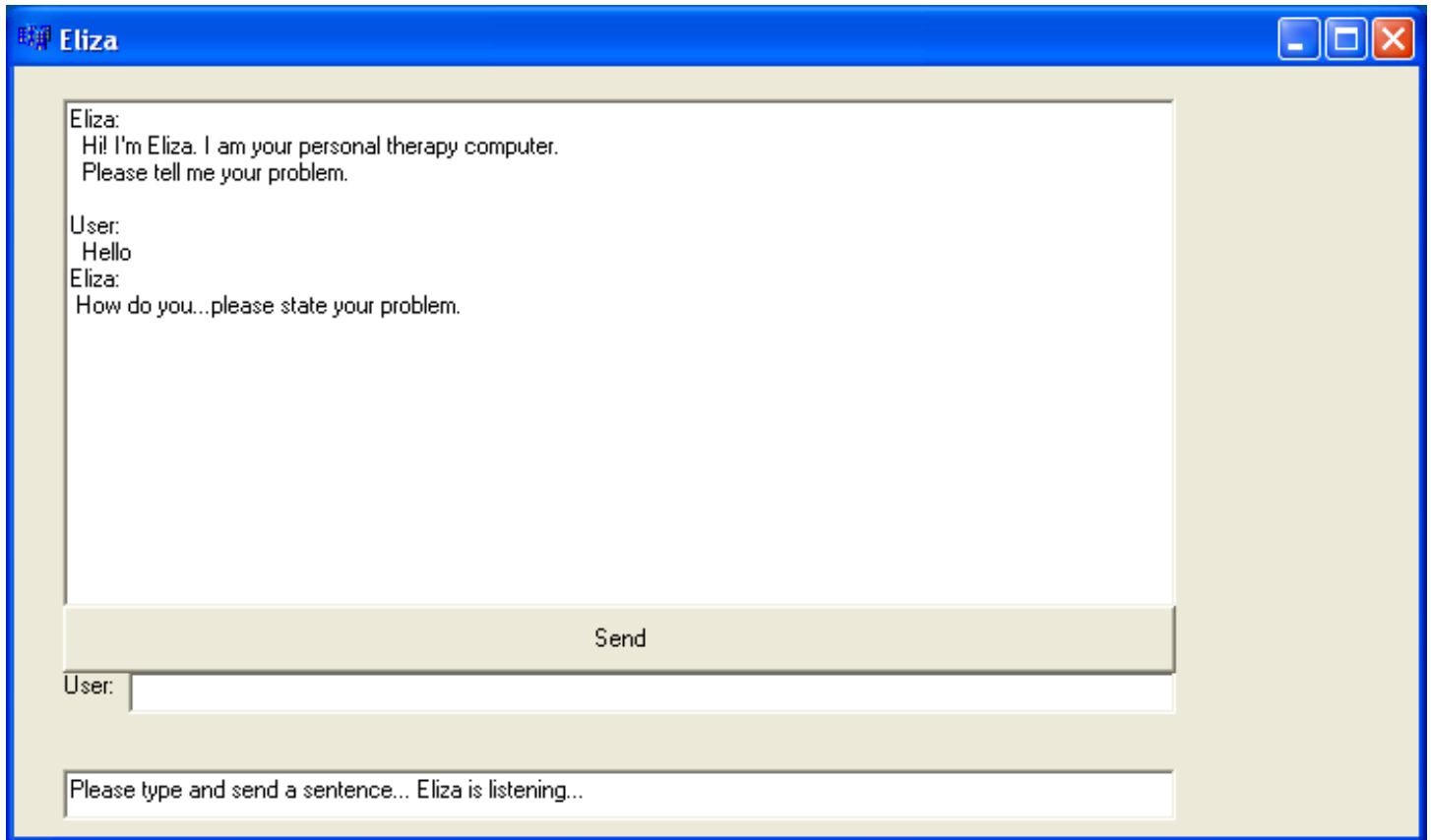


Fig. 4.4.3 Version HEIG-VD.iAi.LaRA d'un chatbot de type Eliza écrit en C++.

4.4.6. Conclusion

Le document présente une brève introduction au logiciel Eliza, un programme simulant de façon grossière un interlocuteur "intelligent". En plus du mode textuel présenté ci-dessus, le programme est capable de générer le message sonore correspondant au texte, ce qui augmente le réalisme et l'impact émotionnel.

Pour plus de détails en ce qui concerne la théorie, on se reportera au polycopié du cours ou aux ouvrages spécialisés.

Pour plus d'information en ce qui concerne l'application concrète, on peut essayer de façon interactive les modules exécutables au laboratoire.

Annexe - Mode d'emploi du programme (en anglais)**Eliza****by Tom Bender**

Enjoy a rousing conversation with your Mac. Eliza will play the part of a psychologist, conducting an open-ended clinical interview. As the patient, you get opinions and advice on a wide range of subjects. Eliza has a large vocabulary and the ability to formulate a virtually unlimited number of responses.

...and if you have Apple's new Speech Manager, you can actually watch and listen as Eliza responds!

Eliza is also a simple, single-window, styled text editor. Eliza is 32 bit clean, System 7 compatible, 68040 cache compatible, virtual memory compatible, big screen compatible and required Apple Event aware. It recognizes extended keyboard keys.

Eliza is ShareWare. The registration fee is \$5. Registered owners are entitled to a copy of the latest source code as well as on-line product support. (See the "Registration" instructions at the end of this manual.)

••• Apple Menu •••

- About Eliza...

This gives a few helpful hints for using Eliza. It also includes instructions on how to contact me for product registration, suggestions or bug reports.

••• File Menu •••

- New

This will open a blank document window. Only one window can be open at a time. The default font and font size for the new window can be set in the Preferences... dialog.

- Open.../Open Any...

This will bring up the standard file selection dialog. Pick the TEXT document you wish to open. If you hold down the shift key while choosing this command, you will be allowed to open the data fork of any file, including non-TEXT files. If you open a TEXT document containing a 'styl' resource (such as documents created by JoliWrite 3.0, Stylus, America Online 2.0 and Tex-Edit), the text will display style attributes (boldface, italics, color, size, etc.). I would discourage you from trying to open active System files, of course.

- Close

This will close the active window. (It is equivalent to clicking in the window's close box.) You will be prompted to save any changes you may have made.

- Save

This will save any changes you have made to the file in the active window. A standard file dialog will be displayed if your document is not yet associated with a disk file.

- Save As...

This will bring up the standard file dialog for creation of a new styled text file on your disk.

- Revert

This will revert to the last saved version of the file in the active window. All changes made since the last time the file was saved will be lost.

- Page Setup

This will display the standard Page Setup dialog allowing you to set certain characteristics for your printer.

- Print...

This will display the standard Print dialog box, which allows you to specify number of copies, print quality, etc.

- Preferences...

This dialog box allows you to customize Eliza's default font and font size. All preferences are stored in a "Prefs" file which is created the first time Eliza is run. The prefs file is usually stored in the System folder or Preferences folder (System 7).

1) Default Font:___ This pop-up menu allows you to change Eliza's default font. The default font is used when a new window is created or when a generic text file is opened.

2) Default Size:___ This pop-up menu allows you to change Eliza's default font size. This size is used when a new window is created or when a generic text file is opened.

- Quit

This will close the Eliza application. You will be prompted to save any altered or unsaved open documents before Eliza quits.

••• Edit Menu •••

- Undo

Undo can reverse your most recent editing action or style change. After you undo something, it can be redone if desired. Undo does not work for the Replace function.

- Cut

Cut will remove the current selection to the clipboard, making it available for pasting to Eliza or other programs. The clipboard holds the text as well as its 'styl' information, making styled selections available to other programs that support standard Mac conventions.

- Copy

Copy will make a copy of the current text selection and place it in the clipboard.

- Paste

Paste will move a copy of the clipboard contents to the current insertion point, deleting any text which may be selected in the active window.

- Clear

Clear will erase the current text selection.

- Select All

All the text in the file will become hilited (selected).

••• Search Menu •••

- Find/Replace...

This dialog lets you enter a text string for which to search. You may also enter a text string to use as a replacement when the search string is found. Selecting the "Find" button will hilit the first occurrence of the search string. (You may then use the "Find Next" or "Replace & Find Again" menu commands.) If you push the "Replace All" button, Eliza will replace all occurrences of the search string with the replacement string, then report how many replacements were made. "Replace All" is not undo-able. The three small buttons labelled "CR", "Tab" and "LF" allow you to easily enter a carriage return, tab or linefeed character into the find string or replace string.

All searches start at the top of the document, unless you select the "Start search at cursor position" option. The "Cases must match" option forces Eliza to look for exact matches. Turn this switch off if you wish case to be ignored. The "Whole words only" option will limit matches to distinct whole words (i.e. looking for 'the' will find 'the' but not 'them').

- Find Again

This option becomes available once a match has been found. It allows you to keep searching down through the document looking for the next match.

- Replace & Find Again

This will replace the hilited word with the replacement string (from the "Find/Replace..." dialog box) and continue searching for the next match.

- Enter Selection

This will copy the current text selection (up to 255 characters) into the "Find/Replace..." dialog box. This text will thus be used for the next "Find Again" command.

- Word Count...

This command shows the word count, line count and character count for the active document.

••• Font Menu •••

The font menu lists all the fonts available to Eliza. The current selection's font is checked.

••• Size Menu •••

The size menu lists a selection of common font sizes. The current selection's font size is checked.

••• Style Menu •••

This is a standard font style menu.

••• Eliza Menu •••

• Start Interview

This command will initiate the conversation between you and Eliza (a.k.a. Mac). First, you will be asked to enter your name. Then Eliza will guide you through the deepest recesses of your soul, in search of true inner peace and harmony. In other words, it will display a statement or question and then wait for you to type your response. Follow each sentence with a carriage return. Please enter your response at the end of the text. Of course, it helps if you use English and only enter a single sentence at a time. Eliza works best when you avoid terse responses and really try to "open up".

This command changes to "Stop Interview" while Eliza is active. It will cause Eliza to bring the session to a close.

• Read Starting At Cursor

This command will cause the active window to be read aloud starting at the current cursor position (or the end of the current selection, if some text is selected). Speech then continues until the end of the document is reached or "Stop Reading" is chosen.

• Read Selection

If some text is selected, this command will read it aloud. It can be stopped or paused as desired.

• Pause/Resume

If text is being read aloud, this will allow you to temporarily pause or restart the reading session.

• Turn Speech On/Off

This command activates the Speech Manager system. It must be selected before any other speech-related commands are usable. While Eliza is speaking, the face at the top of the window will become animated. (If you enter text at the end of the document followed by a control-return, the entered line of text will be spoken if speech is on.)

• Speech Settings...

This command brings up a dialog which allows you to modify some of Speech Manager's speech characteristics. The "Voice:" pop-up menu lists all the voice resources currently available to your system. Selecting a voice will adjust the pitch and rate to that voice's default setting. Try experimenting with different rates and pitches to get the most pleasing results. The "Sample" button will cause a sample sentence to be read aloud.

••• Miscellaneous •••

• Requirements

Eliza requires a Mac or Mac-compatible running System 6.0.7 or better. (I assume it will work with 6.0.3 or better, however I cannot easily test that assumption.) To use the speech facility, you need to install Apple's PlainTalk (Speech Manager). The old Macintosh is no longer supported. Eliza runs fine without the Speech Manager.

• Extended Keyboards

Eliza makes full use of Apple's extended keyboard including the forward delete, home, end, page up, page down and cursor keys. The escape key can be used to cancel any dialog box (just like command-period).

• Other Features

Eliza is 32 bit clean, required Apple Event aware, System 7 compatible, big screen compatible, 68040 cache compatible and adheres to all of Apple's interface guidelines (as best I can tell). Undo is implemented for most functions. If you triple click on a sentence, the entire sentence will be selected. Hold down the shift key during cursor key movements to extend a selection.

• File Formats

Eliza saves its documents in a format which can be read by any text editor or word processor (type 'TEXT'). It saves the style information (font, size, boldface, etc.) in the file's resource fork. Since this style information is ignored by other text editors, they are unable to display anything other than plain text when editing Eliza's files. When Eliza re-opens a file which has been altered by another editor, all the style information will be gone, unless the other text editor also stores the document's style information in the resource fork. For example, America Online 2.0, JoliWrite 3.0, Tex-Edit and Stylus create styled text files which can be easily opened by Eliza

with all style information intact! (Eliza scans the document file for a resource of type 'styl'. The first one it finds is used to style the text. If the document is using 'styl' resources for some other purpose, the text will look funny.)

In order to move a styled Eliza document into another word processor without losing style information, you can use the clipboard to copy and paste the entire document. (Some expensive word processors apparently do not support Apple's standard clipboard formats and may lose the 'styl' information. Go figure...)

Eliza can open any 'TEXT' document (up to 32K) and can also display the data fork of any other file. Eliza can correctly open stationery pad documents.

- Limitations

Eliza is limited to 32K text files. Only a single document window can be active at any time.

- Registration

Eliza is ShareWare. The registration fee is \$5 (US). (Outside the US, you may send an appropriate amount of your local currency, if you are unable to obtain US currency.) Please send your registration fee to:

Tom Bender
5313 Beverly Drive
San Angelo, Texas 76904

- Source Code

Registered users may obtain a copy of the Metrowerks Pascal source code by sending a disk and a self-addressed, stamped envelope to the above address. The source code is intended for your personal use.

- On-Line Support

If you have questions, suggestions or bug reports, you can also reach me via modem:

America Online: TomBB
eWorld: TomBB
Internet: tombb@aol.com

- Other Products

Please also try Tex-Edit Plus (my full-featured text editor), Re-Pete (a memory game) and Azile (Eliza with an attitude), hopefully available on a BBS near you!

Eliza is copyright © 1993-1996 by Tom Bender. All rights reserved.

4.5 Automates en situation ("insectes artificiels")

Une tendance en I.A. est de simplifier au maximum les tâches de réflexion. Alors que traditionnellement, les techniques se basaient sur une représentation interne du monde par des modèles abstraits plus ou moins complexes, l'approche nouvelle consiste à immerger le système "intelligent" dans le monde lui-même. En un mot : le modèle ... c'est le monde réel (cf. Rodney Brooks, MIT).

Il s'agit essentiellement de générer des signaux moteurs, sur la base de signaux captés (perception). Dans le cas extrême, on s'interdit même toute mémoire.

Une source inépuisable d'inspiration est fournie par les systèmes biologiques élémentaires du point de vue cognitif: les insectes. D'où le développement de l'éthologie.

4.5.1 Navigation avec un système sans mémoire; stratégie déterministe.

Table 4.5.1 Deterministic navigation in a labyrinth

<p>Initial conditions: Given particular (simulated) maze, without inner loops Position of starting point for mobile robot.</p> <p>Process Iterate as follows until exit gate is reached: Move ahead, following right wall.</p>
--

Such labyrinths are good examples of situations where solutions can be found by memoryless CS that implement a simple deterministic method.

Fig.4.5.1 Labyrinth with various paths (black: min. solution; dark grey: visited deadends)

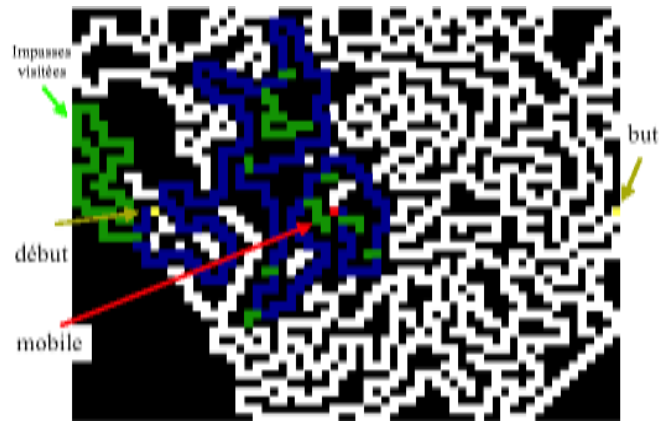
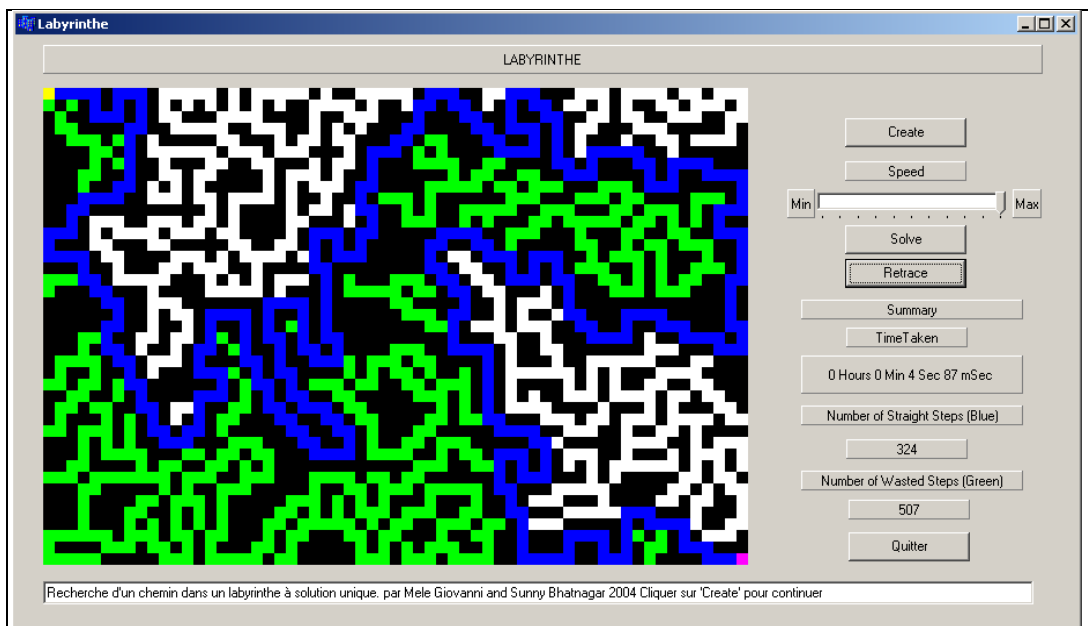


Fig.4.5.2 Labyrinth with various paths (black: min. solution; dark grey: visited deadends)



4.5.2 Fourmi - un animat ou "automate en situation"

Un exemple d'automate en situation très simple, sans mémoire, est fourni par un programme fait au laboratoire par Allegrini et Mantanus: Fourmi.

L'insecte décide de ses actions de la façon la plus simple possible. En particulier, on ne s'autorise aucune mémoire, ce qui empêche toute planification d'actions. Les décisions se prennent à chaque instant.

Essentiellement, l'algorithme est le suivant:

- En principe, c'est-à-dire sur terrain libre, l'"insecte" se déplace en ligne droite vers son but.
- S'il butte sur un obstacle, l'insecte le longe dans la direction le rapprochant de l'objectif.
- Au cas où il est bloqué, c'est-à-dire que le déplacement direct n'est pas possible, et que de longer l'obstacle tant d'un côté que de l'autre l'éloigne du but (la distance se calcule "à vol d'oiseau" - distance Euclidienne), il saute à un endroit plus ou moins proche, selon une loi stochastique qui avantage les petits sauts.

Fig.4.5.3 L'insecte est représenté par un carré accompagné d'une flèche indiquant la direction de mouvement instantané. "X" désigne le point d'où il est initialement parti. Le carré entouré d'un cercle désigne le but. Enfin, les petits carrés sombres désignent les endroits vers lesquels il a temporairement essayé de se déplacer afin d'éviter les obstacles

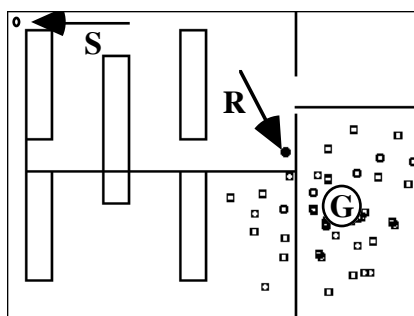
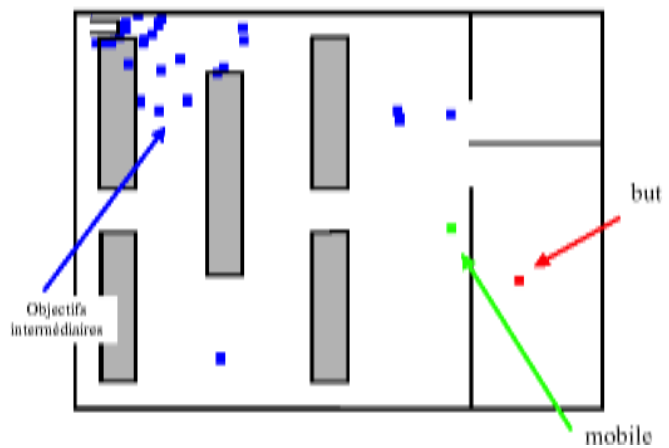


Fig.4.5.4 Robot (R) temporarily blocked in a corner. Subgoals randomly spread around goalsite (G). S:starting position.

6	5	4	3	2	3	4
5	4	3	2	1	2	3
6	5		1	0	1	2
7	6		2	1	2	3
6	5	4	3	2	3	4

Fig.4.5.5 World map with accessibility indexes (values 0 to 7). The goal is grey, and obstacles are black.

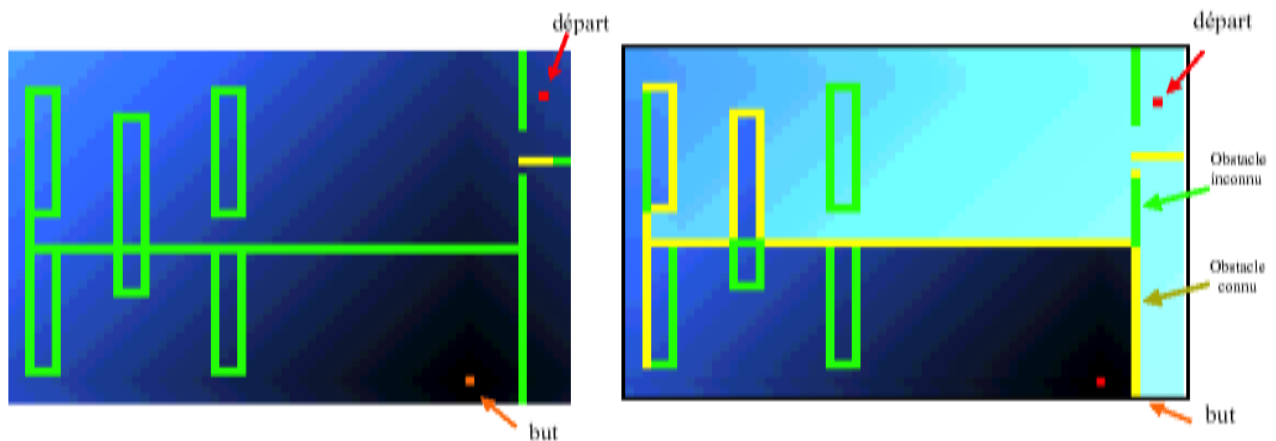


Fig.4.5.6 World map with accessibility indexes (values 0, dark and close to 255, cyan and far). Distances are computed as a wave spreading around obstacles.

Initial conditions:

Given particular (simulated) world, featuring obstacles (walls, tables) represented as a picture on the screen

User-defined positions of final goal, and starting point for mobile robot ("ant").

Process

Iterate as follows until final goal is reached:

Move towards current goal (final goal or short-term self-defined subgoal)

If an obstacle is met then

Follow it and keep moving towards current goal

If blocked in a local minimum then

Set a random subgoal as new value of current goal

If subgoal is reached then

Set final goal as new value of current goal

Table 4.5.2 Stochastic strategy for goal-oriented navigation among obstacles

4.6 Inférence Bayésienne

Une approche ancienne mais toujours actuelle en IA consiste à estimer la vraisemblance d'hypothèses sur la base de probabilités multiples et conjointes. Voyons ceci en 4 points : Introduction ; loi de Bayes ; évidence ; et exemple.

Inférence Bayésienne (1 de 6)

4.6.1 Introduction

On nomme inférence bayésienne la démarche logique permettant de calculer ou réviser la probabilité d'une hypothèse, sachant un fait donné. Cette démarche est régie par l'utilisation de règles strictes de combinaison des probabilités, desquelles dérive le théorème de Bayes. Dans la perspective bayésienne, une probabilité n'est pas interprétée comme le passage à la limite d'une fréquence, mais plutôt comme la traduction numérique d'un état de connaissance (le degré de confiance accordé à une hypothèse).

Notation: Ou : \cup ; Et : \cap ; Sachant : $|$.

4.6.2 Loi de Bayes - Probabilités et Théorème

Il existe seulement deux règles pour combiner les probabilités, et à partir desquelles est bâtie toute la théorie de l'analyse bayésienne. Ces règles sont les règles d'addition et de multiplication.

La règle d'addition $p(A \cup B|C) = p(A|C) + p(B|C) - p(A \cap B|C)$

La règle de multiplication $p(A \cap B) = p(A|B)p(B) = p(B|A)p(A)$

Le théorème de Bayes peut être dérivé simplement en mettant à profit la symétrie de la règle de multiplication

$$p(A|B) = p(B|A)p(A)/p(B).$$

Le théorème de Bayes permet d'inverser les probabilités. C'est-à-dire que si l'on connaît les conséquences d'une cause, l'observation des effets permet de remonter aux causes.

4.6.3 Evidence

Dans la pratique, quand une probabilité est très proche de 0 ou de 1, il faut observer des éléments considérés eux-mêmes comme très improbables pour la voir se modifier. On définit l'évidence par: $Ev(p) = \log(p/(1-p)) = \log(p) - \log(1-p)$. Pour mieux fixer les choses, on travaille souvent en décibels (dB), avec l'équivalence suivante : $Ev(p) = 10 \log_{10}(p/(1-p))$. Une évidence de -40 dB correspond à une probabilité de 10^{-4} , etc. Si on prend le logarithme en base 2, $Ev(p) = \log_2(p/(1-p))$, l'évidence est exprimée en bits. On a $Ev_{dB} = 3,0103 * Ev_{bits}$ ($10^{-3} = \text{env. } 2^{-10}$). L'intérêt de cette notation, outre qu'elle évite de cette manière d'avoir trop de décimales au voisinage de 0 et de 1, est qu'elle permet aussi de présenter la règle de Bayes sous forme additive : il faut le même poids de témoignage (weight of evidence) pour faire passer un évènement d'une plausibilité de -40 dB (10^{-4}) à -30 dB (10^{-3}) que pour le faire passer de -10 dB (0,1) à 0 dB (0,5), ce qui n'était pas évident en gardant la représentation en probabilités.

Evidence (Table d'équivalence) :

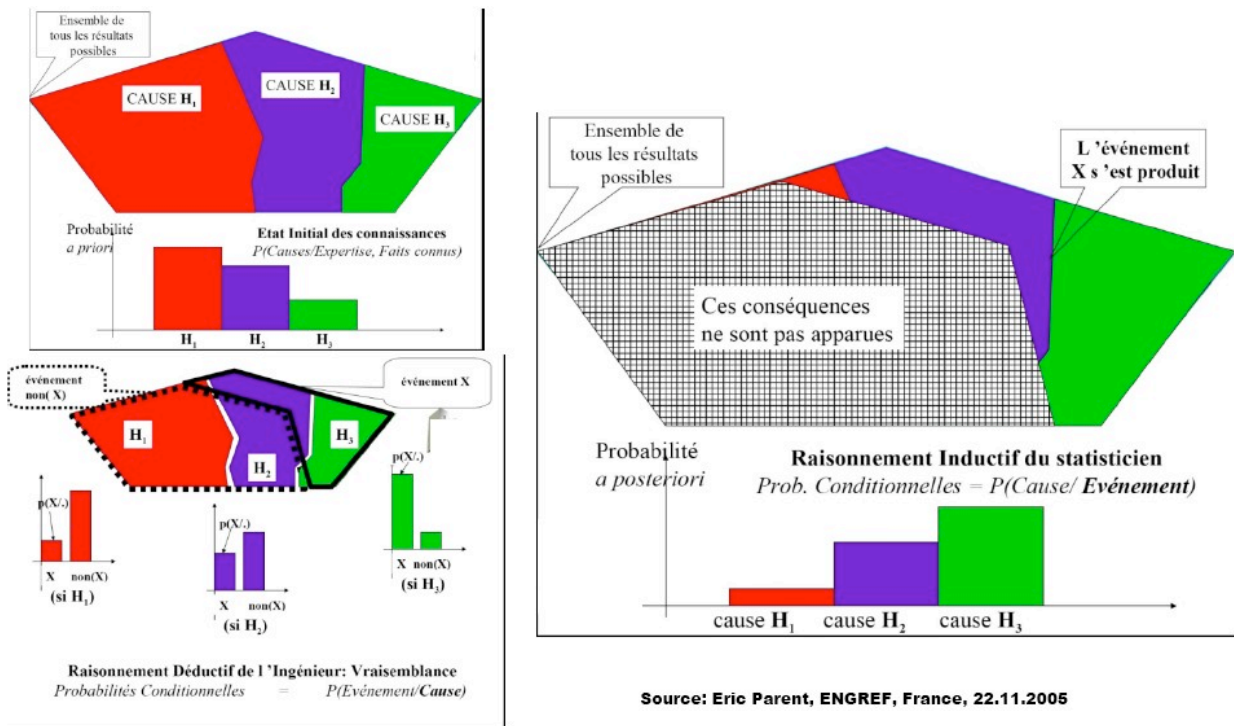
Probabilité	Évidence (dB)	Évidence (bits)
0,0001	-40,0	-13,3
0,0010	-30,0	-10,0
0,0100	-20,0	-6,6
0,1000	-9,5	-3,2
0,2000	-6,0	-2,0
0,3000	-3,7	-1,2
0,4000	-1,8	-0,6
0,5000	0,0	0,0
0,6000	1,8	0,6
0,7000	3,7	1,2
0,8000	6,0	2,0
0,9000	9,5	3,2
0,9900	20,0	6,6
0,9990	30,0	10,0
0,9999	40,0	13,3

4.6.4 Exemple

Exemple: D'où vient la boule blanche?	
Boîte 1, 10 boules noires, 30 blanches	
NNoires1	10
NBlanches1	30
PrDsachantH1	0.75
Boîte 2, 20 noires, 20 blanches	
NNoires2	20
NBlanches2	20
PrDsachantH2	0.5
Probabilité a priori (Prior prob.) de prendre:	
H1: l'hypothèse « On a pris dans la première boîte. »:	
PrH1	0.5
H2: l'hypothèse « On tire dans la seconde boîte. »:	
PrH2	0.5
D: (données, data) On tire une boule blanche	
PrH1sachantD?	
$P(H1 D) = \text{Num}/\text{Denom}$	
Num = $(P(D H1)*P(H1))$	
Num	0.375
Denom= $P(D)$	
$P(D)=P(D H1)*P(H1)+P(D H2)*P(H2)$ (intégrale toutes hypothèses)	
PrD	0.625
Denom	0.625
PrH1sachantD	0.6

Table 4.5.2 Programme Excel HEIG-VD.iAi.LaRA, selon référence http://fr.wikipedia.org/wiki/Inf%C3%A9rence_bay%C3%A9sienne

4.6.5 Illustration graphique de la méthode de Bayes



HESSO.HEIG-VD, J.-D. Dessimoz, 05.04.2011

40

Fig. 4.6.5 Méthodes de Bayes sous forme graphique (extrait de la présentation powerpoint du cours AIC)

5. LOGIQUE FLOUE

Le génie logiciel et les techniques modernes de traitement de signal nous a progressivement fait passer de l'analogique, continu, au numérique, discret, en particulier Booléen. Un domaine particulier de l'IA a consisté, en réaction à cette tendance, à revenir vers des possibilités fines de description des objets, les techniques dites « floues ». Après une présentation générale, deux applications concrètes seront considérées, l'une avec le logiciel Togai Fuzzy Logic, et l'autre réalisées au laboratoire LaRA de la HEIG-VD.

5.1 Présentation générale de la logique floue

La logique floue a été proposée il y a près de 20 ans par Lofti Zadeh. L'idée fondamentale était de dépasser les deux niveaux typiques de la logique classique ou booléenne. Par exemple, en logique on est jeune ou on ne l'est pas. Or Zadeh propose d'ajouter à chaque notion ("jeune" dans notre exemple) une fonction d'appartenance à multi-niveaux, voire continue. On est ainsi plus ou moins jeune, avec a priori n'importe quel degré possible.

Certains critiquent l'approche de Zadeh: En fait lorsqu'on s'échappe de la logique, on tombe par définition dans le monde analogique. Il n'y aurait donc rien d'original. Soit la fonction d'appartenance atteint l'un parmi un nombre fini de niveaux, et on peut le coder en binaire - le monde logique. Soit elle est continue, et tombe alors dans le monde classique de l'analogique.

En pratique, les systèmes à logique floue d'aujourd'hui permettent de combiner les avantages des deux mondes: logique et analogique. Par rapport au monde de la logique, il y a progrès en ce que les caractéristiques sont maintenant considérées à plus que deux niveaux. Par exemple on est "très jeune", "moyennement jeune", "un peu jeune", etc. Et par rapport au monde analogique, on peut fortement augmenter la complexité des structures de traitement, en particulier en termes de cascade d'opérations successives. Les structures, reprises du monde de la logique, prennent typiquement l'aspect d'ensembles de règles de production, avec interrelations multiples, à la manière des systèmes-experts.

L'exemple présenté ci-après, traitant d'une servocommande pour moteur, est tiré de [Kap]. Le régulateur à logique floue reçoit deux mesures numériques: "newerror", l'erreur courante de position, et "derror", pratiquement sa dérivée puisqu'il s'agit de la différence entre l'erreur courante et la précédente. La valeur de sortie de la commande, "out_val", qui est ultérieurement convertie en tension pour le moteur, est tirée d'une base de règles - un algorithme (présenté ici sous forme de table) qui définit la relation entre variables d'entrée et de sortie. Résidente en mémoire, la base de règles a des groupes de fonctions d'appartenance floues, chacune représentant une variable (positif grand (PG), positif petit (PP), nul (Z), négatif petit (NP), et négatif grand (NG)) - au lieu de "0" et de "1". Par exemple, l'une des commandes est "si derror est Z et newerror est PG, alors out_val est PG".

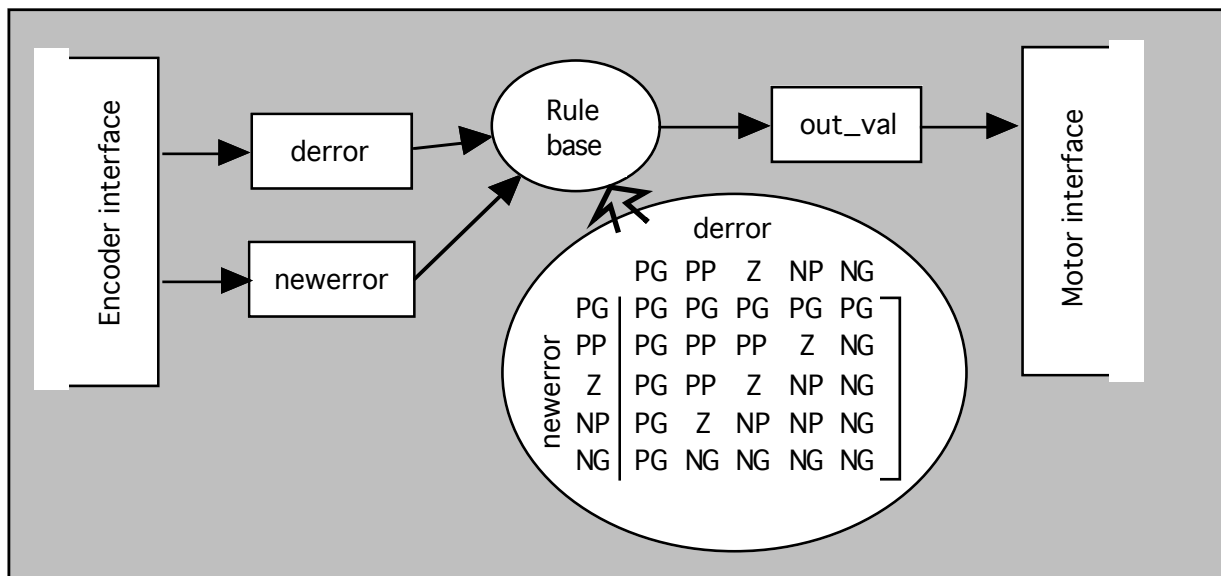


Fig. 5.1 Commande de moteur à logique floue (d'après Togai Infralogic Inc.)

Remarques et commentaires:

- L'exemple est simple en ce sens qu'un seul niveau de traitement sépare les signaux de sortie de ceux d'entrée. Sans probablement jamais être très nombreux, en pratique il arrive tout de même qu'il y en ait quelques uns de plus.
- La fonction d'appartenance à l'un des niveaux (par exemple "PP") est généralement trapézoïdale par rapport à la variable continue sous-jacente (voir fig. ci-dessous). Ainsi, en supposant que l'erreur corresponde fondamentalement à un signal analogique variant entre -10V et +10V, on aurait par exemple un segment de droite valant 0 entre -10V et 0V, puis un segment de droite de pente 1/2 entre 0V et 2V, puis un segment constant valant 1 jusqu'à 4V, suivi d'un segment de pente négative -1/2 jusqu'en 6V et finalement 0 depuis là jusqu'en 10V.

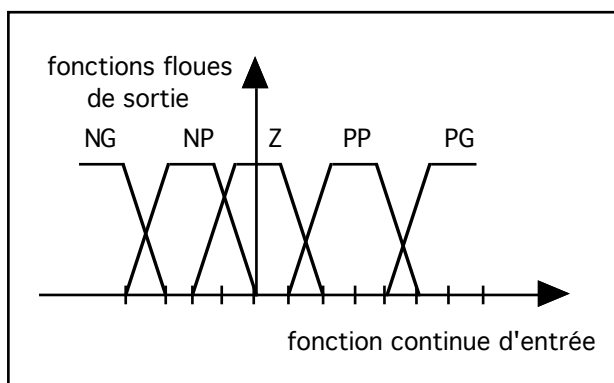


Fig. 5.2 Les variables floues de la commande pour moteur présentée ci-dessus ont des fonctions d'appartenance continues

- Dans l'exemple, l'équivalence entre règle de production -typique des systèmes experts-, et table de conversion - typique des systèmes logiques câblés - est explicite. On peut évidemment faire un pas de plus et constater que cela pourrait aussi prendre la forme d'un microprogramme, ou d'instructions Pascal... La théorie introduite plus haut dans ce chapitre a le mérite de n'impliquer aucune distinction et de couvrir aussi bien chacun de

ces modes de réalisation. Il en ressort en particulier que la quantité de connaissance mise en jeu ici est très faible (elle est de l'ordre d'une dizaine de [lin]).

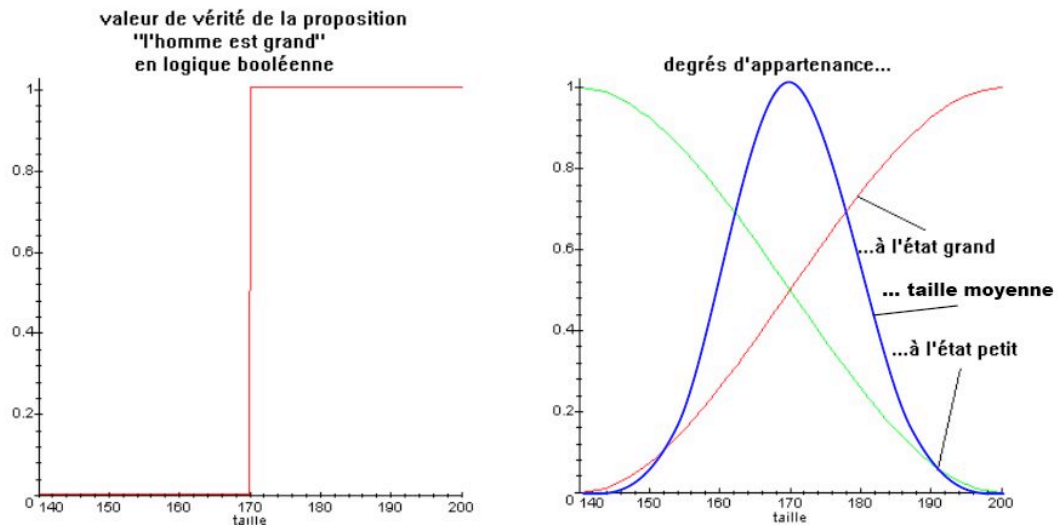


Fig. 5.3 **Cas de la logique booléenne** **Cas de la logique floue**

En résumé, les concepts de base de la logique floue sont les suivants:

- Les fonctions d'appartenances, qui affinent les attributs Vrai ou Faux d'une variable logique, en faisant donc une variable floue;
- Les opérateurs Min et Max, qui généralisent les Ou et Et de la logique.

5.2 Togai Infralogic - systèmes à logique floue

Togai Infralogic propose divers produits pour le développement et l'implémentation industrielle de systèmes à logique floue. Cela va de didacticiels élémentaires à des cartes numériques spécialisées, en passant par des environnements puissants pour le développement de systèmes nouveaux.

Dans la figure ci-dessous, un système de démonstration met en évidence le rôle des éléments principaux d'un système à logique floue.

Il s'agit du problème de l'équilibre d'un pendule renversé. Divers paramètres tels que le poids du mobile, la puissance du moteurs, l'action de perturbations, etc. peuvent interactivement varier.

Des paramètres internes, tels les règles de régulation ou les courbes d'appartenance des variables logiques peuvent aussi se modifier et apparaissent de façon très claire sous forme graphique.

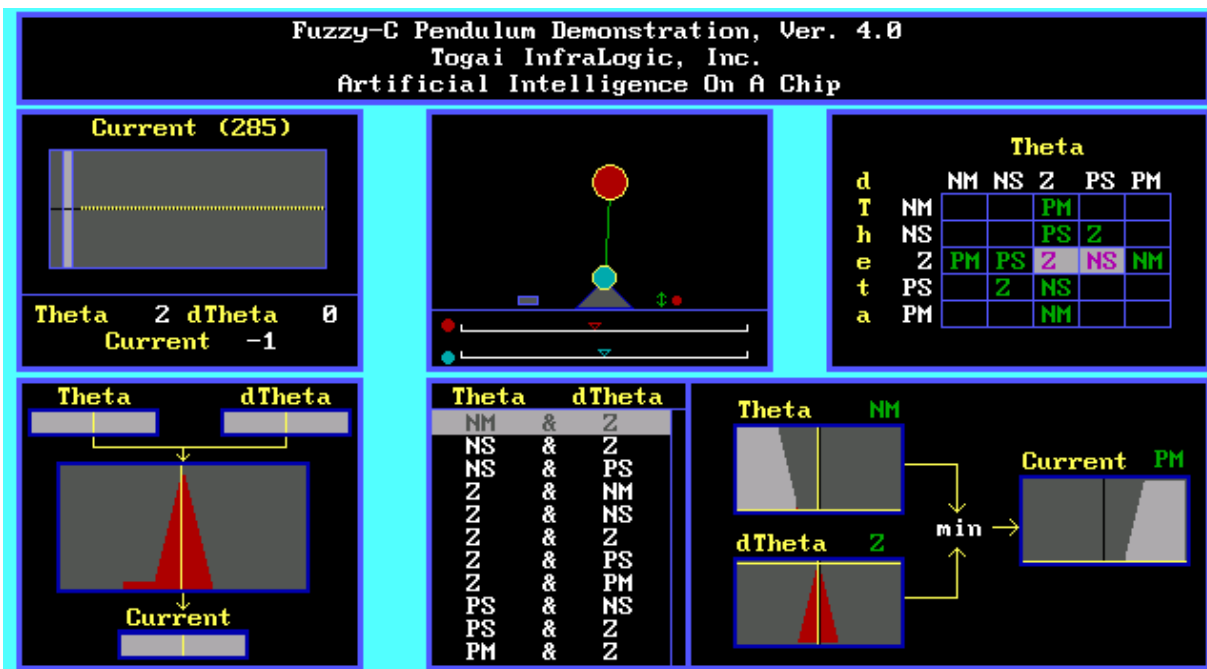


Fig. 5.4. Vue d'ensemble du système de commande à logique floue pour maintenir l'équilibre d'un pendule renversé.

5.3 Réalisation HEIG-VD - système à logique floue pour la régulation d'un balancier

La première version était en Pascal, puis une version a été réalisée en C++.

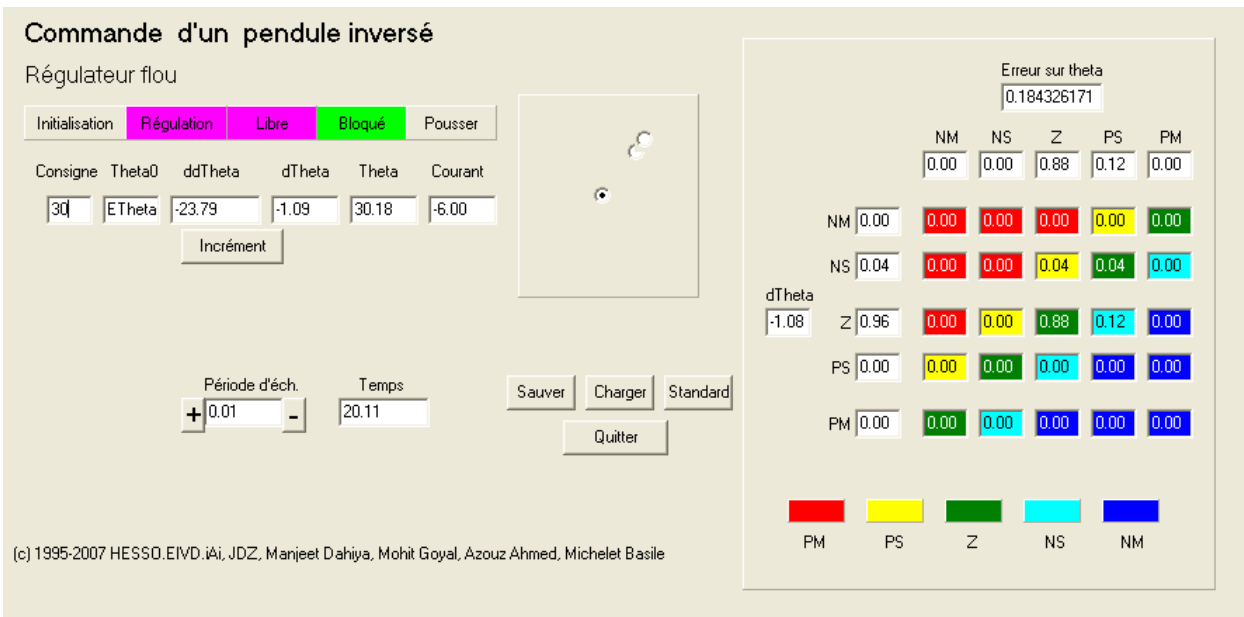


Fig. 5.5 Ecran de commande et pendule inverse commandé par logique floue (Fig. Guerdat-Gumy 2010).

6. RESEAUX NEURONAUX

Une technique apparemment très proche pour implémenter l'IA se base sur une modélisation de l'infrastructure nécessaire à l'intelligence chez l'humain, les neurones. Après une vue générale de cette technique, des cas particuliers sont développés, au LaRA et dans le monde.

6.1 Présentation générale des réseaux neuronaux

Pour aller plus loin dans le sens des performances en traitement d'information, les processeurs individuels d'aujourd'hui ne suffisent plus. Il faut recourir - comme c'est le cas dans le cerveau humain - à un grand nombre de processeurs agissant en parallèle.

L'idée resurgit donc (elle était apparue dans les années 50, puis avait été écartée...) de concevoir des systèmes inspirés des neurones.

En plus de caractéristiques avantageuses en termes de rapidité de calcul et de robustesse (faible sensibilité aux pannes et erreurs ponctuelles), les systèmes actuels à réseaux de neurones apportent des solutions élégantes dans certaines applications: séparation de signaux mélangés, tels que des conversations bruitées (Hérault-Jutten), structuration sans tuteur (Kohonen), coloration de cartes géographiques (modèles compétitions-coopérations), reconnaissance par corrélation (Hopfield), régulation de systèmes sans modèle explicite (Rétropropagation).

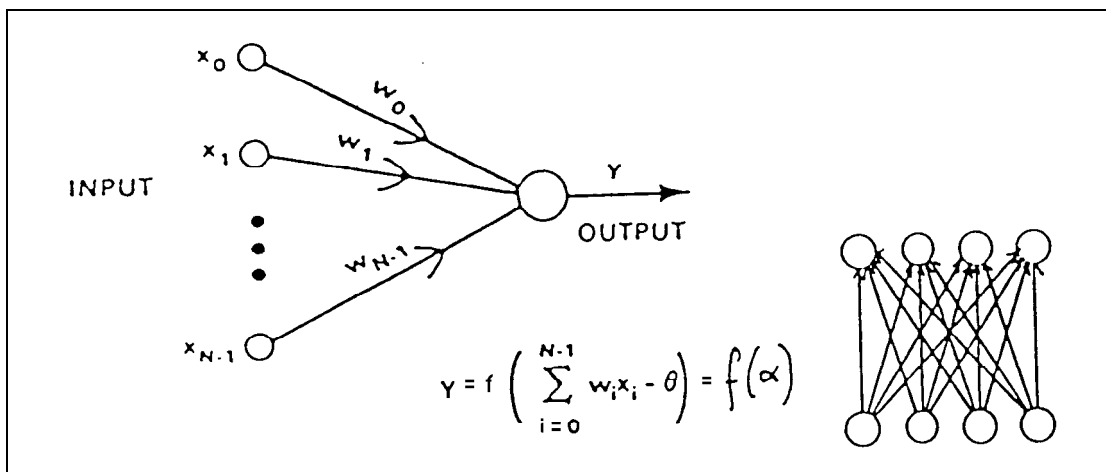


Fig. 6.1.1 A gauche, un noeud (neurone) reçoit des informations pondérées provenant de divers autres noeuds. A droite, on voit un réseau organisé en deux couches.

Les systèmes actuels mettent en jeu des centaines de cellules, qui agissent essentiellement en parallèle sur deux niveaux. A mon avis, il n'est pas possible, avec si peu de niveaux, d'atteindre des performances intéressantes en termes de quantité de connaissance ou d'expertise - sauf pour quelques cas particuliers.

Il faut bien voir que les traitements simultanément exécutés sur de tels systèmes parallèles ne peuvent être que simples, voir triviaux. Et si l'on veut faire du ping-pong entre les deux niveaux (c.-à-d. par itération faire partiellement l'équivalent de ce que de nombreuses couches successives permettraient), les dangers d'instabilité sont énormes.

En pratique, la structure - et donc les possibilités - des systèmes neuronaux proposés peuvent se tester qualitativement sur n'importe quel ordinateur. Et il y a peu de résultats, à ma connaissance, qui justifient qu'on accélère ce type de traitement en réalisant des systèmes spécialisés.

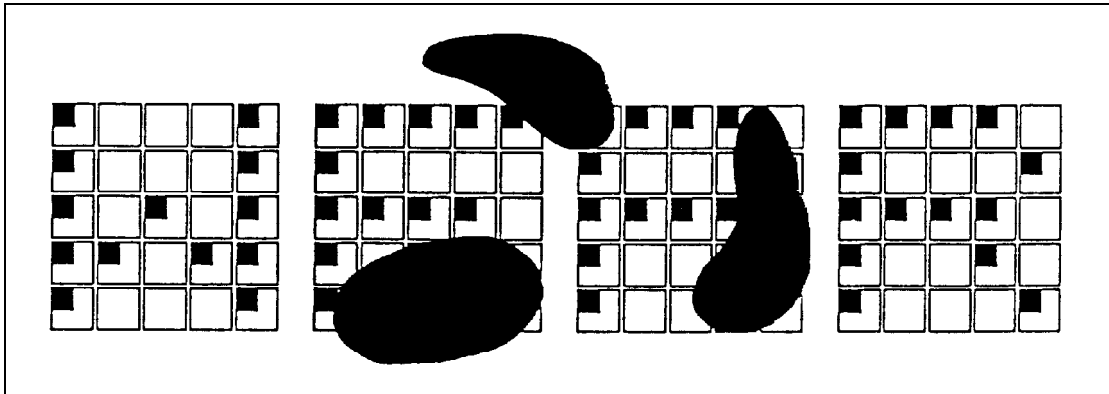


Fig. 6.1.2 Un système neuronal à deux couches a été entraîné pour reconnaître une dizaine de mots-types. Dès lors, il est capable de reconnaître que les pixels cachés correspondent au mot-type "wear" ([Pfeiffer & al., 89]).

En résumé, les concepts de base des réseaux neuronaux sont les suivants:

- Un neurone fait la somme des entrées pondérées par une grandeur spécifique à chaque liaison
- L'apprentissage se fait typiquement en accroissant le poids des liaisons actives lors de situation positives (lien entre une entrée "vraie" et une sortie "vraie" également)
- Le réseau "généralise" en ce sens qu'il délivre une sortie aussi pour les configurations d'entrée qui ne sont jamais apparues dans la phase d'apprentissage.

6.2 Réalisation HEIG-VD - système à réseau neuronal pour la régulation d'un balancier

Résumé

Le document disponible au laboratoire ou sur le réseau de l'HEIG-VD contient des indications visant à préparer une manipulation ambitieuse au Laboratoire de Robotique et d'Automatisation de l'HEIG-VD: la régulation par réseau neuronal d'un pendule inversé. Il s'agit là d'une simulation avec affichage graphique du pendule et un ensemble de commandes interactives.

On rappelle d'abord, sur un plan théorique les bases du calcul neuronal. Puis le pont est fait avec le cas concret du pendule inversé.

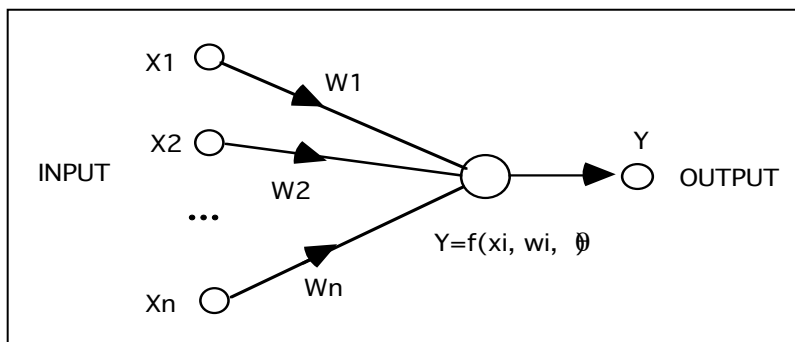


Fig.6.2.1 Modèle de Hopfield pour un neurone

Le document présente aussi sous forme structurée les divers modules de l'environnement pour la commande par réseau neuronal développé au LaRA. Le logiciel tourne sur un ordinateur compatible IBM-PC, et se programme en Pascal. Pour une description détaillée des modules logiciels, le lecteur se reportera au code disponible au laboratoire.

Certains modules ont été développés par des étudiants dans le cadre de miniprojets (en particulier Liardon [1]).



Fig.6.2.2 Régulation d'un pendule inversé par réseau neuronal (ancien doc. Hesso-HEIG-VD-iAi ; premier système avec rendu graphique en temps réel)

6.2.1. Introduction

Le document présente sous forme structurée les divers modules d'un environnement pour la commande par réseau neuronal développé de façon originale au Laboratoire de Robotique et d'Automatisation de l'HEIG-VD. Le code tourne sur un ordinateur compatible IBM-PC, et se programmait en Pascal dans sa version graphique originale, et en C++ actuellement.

En première partie, le document comprend en outre une brève introduction aux réseaux neuronaux.

Certains modules ont été développés par des étudiants dans le cadre de miniprojets (en particulier Liardon [1]). Voir ailleurs une description détaillée des modules logiciels.

D'autres logiciels didactiques pour les réseaux neuronaux sont disponibles au labo. Il s'agit notamment de modules permettant de réaliser des réseaux de Hopfield, de Kohonen, ou de Héroult-Jutten.

6.2.2. Éléments de théorie

Avant de parler de l'application à la commande d'un pendule, on rappelle ici quelques éléments de théorie. Pour une présentation plus fouillée des réseaux neuronaux, le lecteur se reportera au fascicule "Intelligence artificielle et génération de plans d'action" [4], ou à l'un des nombreux livres spécialisés existant sur ce sujet (par ex. [5, 6]).

La commande d'un pendule inversé est un problème classique en régulation, qui a donné lieu à d'innombrables essais. A l'HEIG-VD, il existe notamment une maquette réelle (Prof. Mudry), une solution avec variables d'état, sur MatLab (Profs F. Mudry et M. Etique), un logiciel pour la commande par variables floues, etc.

A Généralités

La notion de "neurone" en tant que modèle biologique pour des systèmes électroniques de calcul a été introduite dans les années 50 (le "Perceptron"). Ce concept a connu des hauts et des bas, une mode lui étant favorable se suivant d'une phase au contraire où ses limites étaient mises en évidence.

Avec les années 90, et les progrès de la micro-électronique qui permettent d'envisager des systèmes richement parallèles, le concept vit une nouvelle période "faste".

Schématiquement, un neurone comporte n liaisons d'entrée, chacune pondérée par un facteur w_j . Dans le cas général, la sortie, y , est une fonction f de la somme pondérée des entrées, après décalage d'un certain seuil θ (cf. fig. 1).

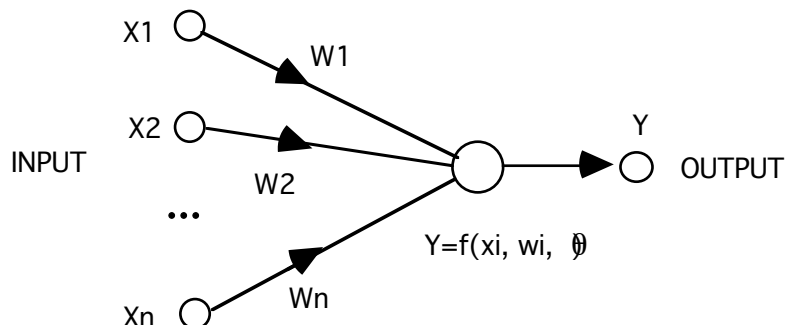


Fig. 6.2.3 Représentation schématique d'un neurone, avec ses n liaisons d'entrée, chacune pondérée par un facteur w_j . La sortie, y , est une fonction f de la somme pondérée des entrées, après décalage d'un certain seuil θ .

Diverses fonctions "f" sont envisageables, pour faire passer des entrées aux sorties. En général, on a tout d'abord une somme pondérée: chaque entrée, dont le niveau peut être booléen ou scalaire, se voit multipliée par le poids de la liaison qu'elle traverse ("synapse"). Ce produit élémentaire va ensuite gonfler la somme générale qui s'opère dans le neurone:

$$s = \sum_{i=1}^n w_i x_i$$

La somme s est ensuite traitée de façon différente suivant les cas. Très souvent, on lui retranche une valeur de seuil, θ , puis on en prend le signe pour obtenir une sortie tout-ou-rien (booléenne). Cf. Fig. 6.2.4. Mais bien d'autres fonctions existent, allant de l'identité à des fonctions de pondération progressive telle f_2 , sur la fig. 6.2.5.:

$$y = f\left(\sum_{i=1}^n w_i x_i - \theta\right)$$

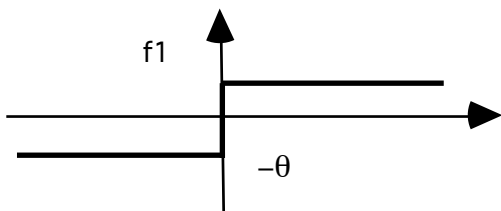


fig. 6.2.4 Fonction binaire "signe"

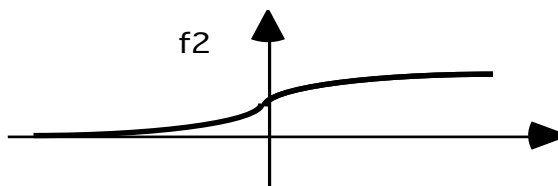


fig. 6.2.5 Fonction de type "courbe en S"

Les neurones sont en principe multiples, côte-à-côte, formant ainsi une "couche". De plus, les neurones sont cascadables, les sorties d'une couche de neurones entrant dans les neurones de la couche suivante. En général, on a 2 ou 3 couches.

Il arrive que les couches rétroagissent. Dans ce cas, certaines sorties de couches aval reviennent influencer des entrées de couches amont. Théoriquement cela est plaisant car un système à 2 ou 3 couches peut parfois fournir un résultat équivalent à celui d'un réseau comportant un très grand nombre de couches successives sans rétroaction. Mais il faut alors compter avec des difficultés sérieuses en termes de garantie de stabilité.

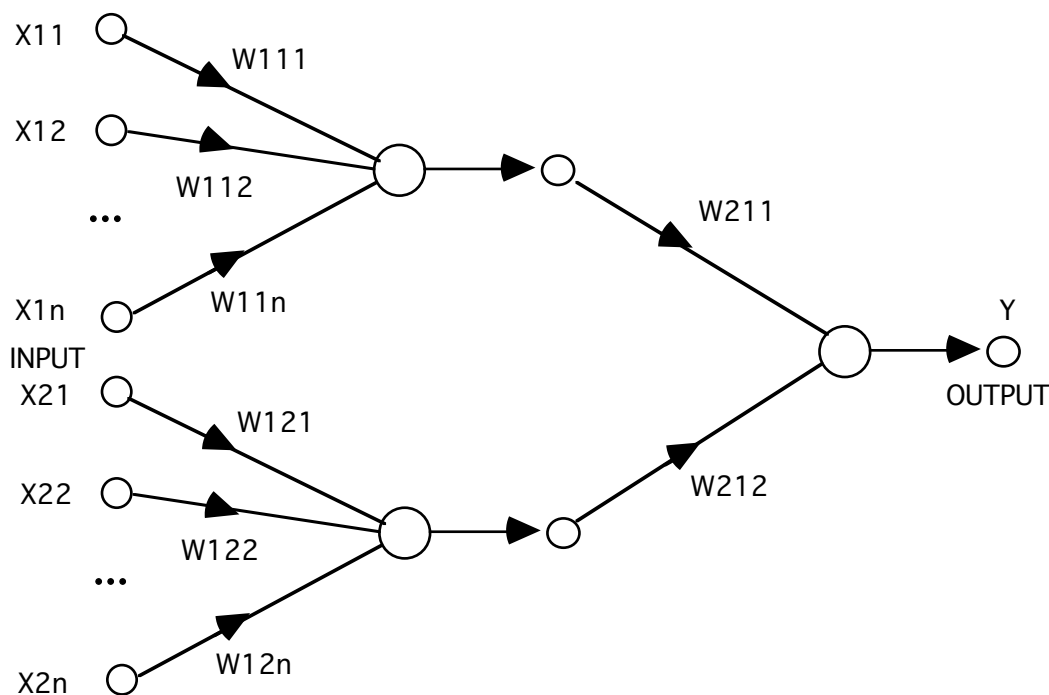


Fig. 6.2.6 Réseau neuronal à deux couches (en général $X_{11}=X_{21}$, mais $W_{111} \neq W_{121}$)

Pour entraîner un réseau, différentes stratégies sont possibles. L'utilisateur peut directement programmer le poids des diverses synapses. Mais on peut aussi parfois laisser le système se programmer de lui-même.

Un cas intéressant est celui de "Hebb". On présente au réseau des associations correctes entrées-sorties. Sur cette base, le programme incrémente le poids des synapses concernées. Par exemple si pour une entrée E_i , on constate que le neurone N_j doit "s'allumer" (c'est-à-dire être actif), alors le poids W_{ij} augmente d'un certain montant α :

$$w_{ij} = w_{ij} + \alpha$$

B Application à la commande d'un véhicule à un carrefour

Voyons d'abord une application élémentaire, qui concerne la commande automatique d'un véhicule à un carrefour. Il est très simple dans ce contexte de pratiquer les différentes notions telles que poids des liaisons, évaluation d'une opération neuronale, apprentissage selon loi de Hebb ou encore décision selon neurone le plus actif, et généralisation (réponse à des cas non appris).

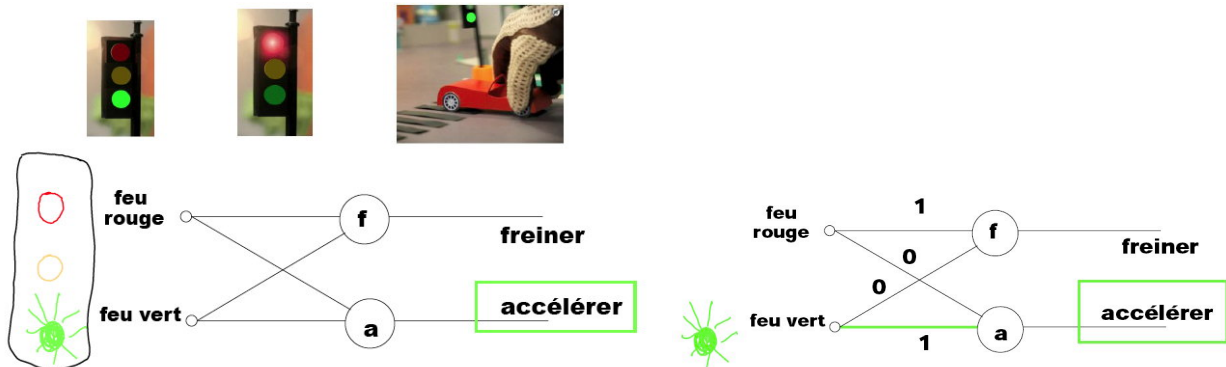


Fig. 6.2.6b Réseau neuronal élémentaire pour la solution de principe du comportement d'un véhicule à un feu de circulation routière.

C Application à la commande d'un pendule

L'application que l'on étudie ci-dessous est relativement complexe et concerne la commande automatique d'un pendule inversé. Il s'agit, malgré l'attraction terrestre, de maintenir un "bâton" en position verticale. Pour cela, on dispose d'un moteur et d'une source de courant.

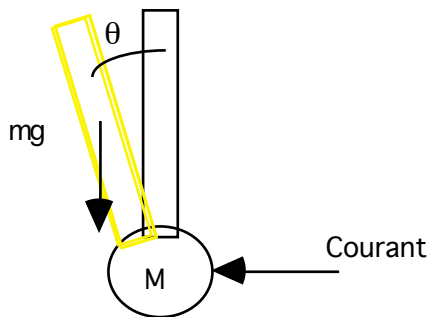


Fig. 6.2.7 Représentation schématique d'un pendule inversé. Le "bâton" est entraîné par le rotor du moteur.

Pour nos essais, on définit 100 signaux booléens d'entrée.

Un premier groupe de 50 représente, avec 2% de précision, l'angle (en régulation classique, on voit implicitement là l'erreur angulaire instantanée; pour faciliter cet artifice, on a fixé le 0° de rotation en position verticale, c'est-à-dire en position de consigne).

Le deuxième groupe de 50 entrées représente la vitesse angulaire, avec une résolution de 2% également (là aussi, en régulation classique on verrait plutôt l'écart entre consigne et mesure pour la vitesse; cela revient pratiquement au même parce qu'ici on a une consigne de vitesse égale à 0°/s). (cf. fig. 6.2.8)

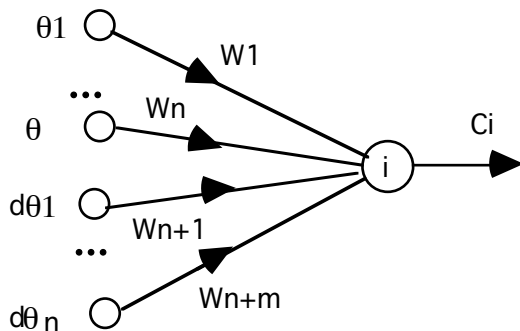


Fig. 6.2.8 Chaque neurone décide d'un niveau de courant donné, en fonction du poids qu'il attribue aux divers niveaux angulaires et de vitesse à l'entrée

On envisage une quinzaine de niveaux différents de courant. Dans ce but, on définit donc 15 neurones, qui tous ont les entrées susmentionnées (cf. fig. 6.2.9).

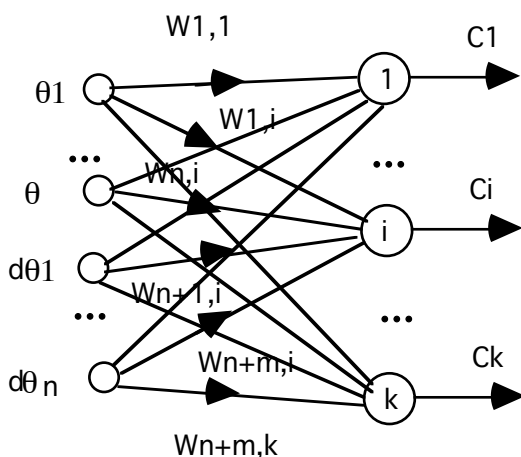


Fig. 6.2.9 Vue d'ensemble du réseau neuronal.

Dans notre exemple, le programme sélectionne finalement le neurone le plus actif pour envoyer au moteur (simulé) le niveau de courant approprié.

Par ailleurs, le logiciel développé au LaRA calcule l'évolution du pendule sur la base des lois physiques (relations entre angle, masses, longueurs, gravité, etc.); il gère aussi la souris; l'affichage graphique dans des fenêtres en relief; le choix de modes de travail par "boutons"; il permet de réguler le pendule, de le bloquer pour analyser les résultats intermédiaires, de le lâcher sans régulation, ou encore de lui donner un petit choc pour le perturber.

On peut enfin par d'autres petits programmes visualiser les poids du réseau, ou encore obtenir le courant de commande pour un couple angle-vitesse librement formulé.

Une variante non mise à disposition consisterait à ajouter une couche neuronale. On ferait alors la somme des sorties neuronales actuelles, pondérées par leur courant nominal (cf. fig. 6.2.10).

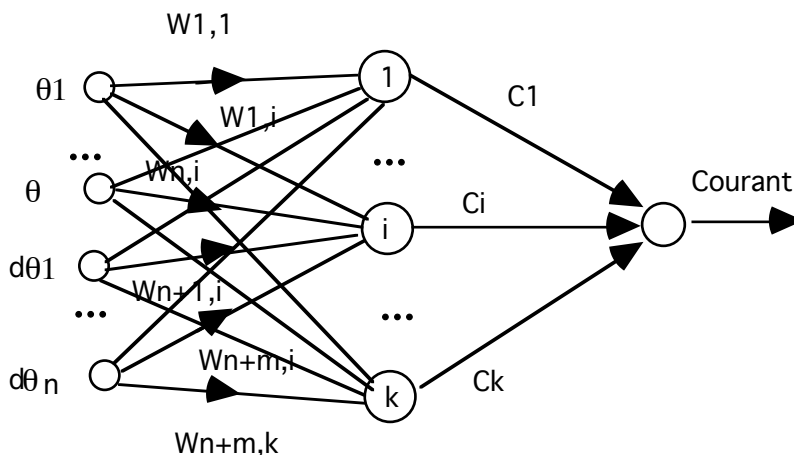


Fig. 6.2.10 Réseau comportant une couche supplémentaire pour faire la somme pondérée des courants choisis par le réseau de la fig. 6.2.9

6.2.3. Vue d'ensemble des modules *Soft-LaRA* pour la régulation par neurones

Le fichier de projet C++ "SimBINR.C++.Train&Regulation 1.7" contient le code source pour la simulation d'un balancier commandé en rotation de façon à être vertical.

Pour réaliser cela, plusieurs modules sont employés, dont particulièrement les suivants, développées à l'HEIG-VD:

- uRegNe.cpp: unité contenant les types, variables et procédures de base pour un réseau neuronal de commande.
- uBal.cpp: unité calculant et affichant l'état d'un balancier, d'après les lois physiques.

En outre, plusieurs programmes sont utiles, à titre accessoire, dans cette application de commande neuronale de balancier:

- GetRef.cpp : calcule des entrées (Thêta, dThêta) et des sorties (Courant) sur la base d'un régulateur Prop.-Dérivé, pour l'entraînement éventuel d'un réseau neuronal. C'est le premier programme à exécuter lors d'une nouvelle installation. Il suffit de l'exécuter 1 fois.
- TrainRN.cpp: Fait l'apprentissage, càd passe des données de référence aux poids du réseau neuronal. C'est le deuxième programme à exécuter lors d'une nouvelle installation. Il suffit de l'exécuter 1 fois.

6.2.4. Programme de simulation

Le fichier SimBNe contient le code source pour la simulation d'un balancier commandé en rotation de façon à être vertical (cf. Annexe pour voir une copie d'écran).

Le mode de régulation retenu ici est celui des réseaux neuronaux.

A. Calcul de valeurs d'entrées-sorties de références, pour l'entraînement

Le programme "GetRef" calcule des entrées (Thêta, dThêta) et des sorties (Courant) sur la base d'un régulateur P-D, pour l'entraînement ultérieur éventuel d'un réseau neuronal.

```

Program GetRef;
uses Crt, uSLGrap6, uBal2, uRegNe2, uRegPD;
Const
  NValeursParVar=50; (* résolution... *)
...
begin
...
  Assign( Ft, 'RegFile.dat' );
...
  InitPendule;
...

```

```

For iTh:= 1 to NValeursParVar do
  For idTh:= 1 to NValeursParVar do
    begin
      (* Définition systématique de valeurs de Theta et dTheta *)

      (* Calcul de valeurs de Theta et dTheta *)
      CommandePD( Theta, dTheta, Courant);
    ...
      (* Ecriture de valeurs d'E/S typiques pour entraînement ultérieur *)
      WriteLn(Ft, Theta:8:0, dTheta:8:0, Courant:8:0 );
      end; (* for *)
    ...
  end.
  
```

B. Apprentissage - entraînement du réseau sur les données de références

Le programme "TrainRN" fait l'apprentissage, càd passe des données de référence aux poids du réseau neuronal

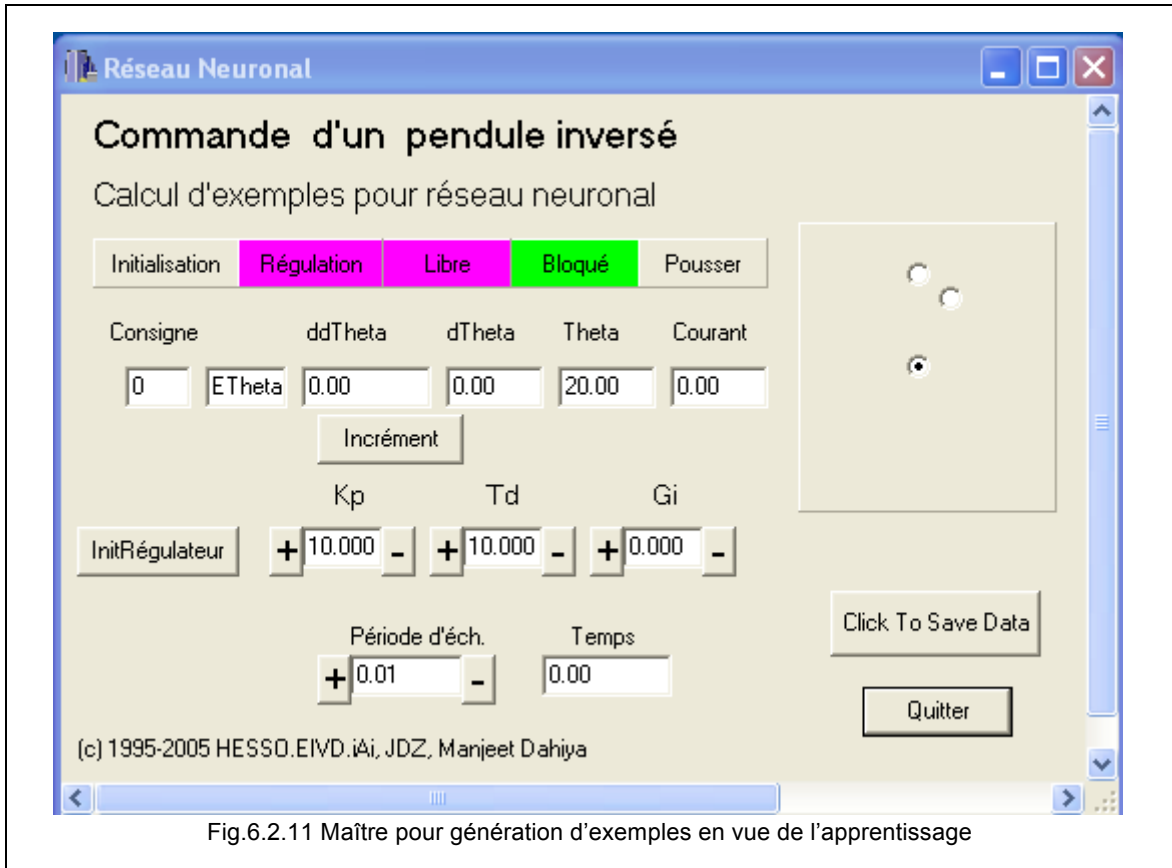


Fig.6.2.11 Maître pour génération d'exemples en vue de l'apprentissage

Extraits des données – Exemples pour apprentissage

Theta	dTheta	Courant
...		
-180.00	795.92	2034.00
-180.00	836.73	2034.00
-180.00	877.55	2034.00
-180.00	918.37	2034.00
-180.00	959.18	2034.00
-180.00	1000.00	2034.00
-172.65	-1000.00	1950.98
-172.65	-959.18	1950.98
-172.65	-918.37	1950.98
-172.65	-877.55	1950.98
-172.65	-836.73	1950.98
-172.65	-795.92	1950.98
-172.65	-755.10	1950.98
-172.65	-714.29	1950.98
...		

6.2.5. Unités spécialisées pour la gestion de réseau neuronal

Extraits de code – Structure

```

/*          / Entrée 1      *)
(*          (var) ...      *)
(*          / (\)...      *)
(*  Neurone      ...      *)
(*  / \ (\)      *)
(* réseau (var)      *)
(* \ \ Entrée N      *)
(*  Neurone ...      *)
(*          */

```

Extraits de code – Processus d'apprentissage

Loi de Hebb: renforcement des liaisons actives au deux bouts, selon les exemples fournis

```

void ModifPoids(NReseau &N, int i, int j, int k)
{
    // N: No du neurone;
    // i: No de la variable;
    // k: No de l'entrée active
    float PoidsTemp;
    PoidsTemp=N[i-1].Poids[(j)*NEntreesParVar+k];
    PoidsTemp = PoidsTemp + alpha;
    N[i-1].Poids[(j)*NEntreesParVar+k]=PoidsTemp;
}

```

*Extraits des données – Exemples de réseau**Neurone No 10*

```

0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 50.00 50.00 50.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 15.00
60.00 60.00 15.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00
0.00 0.00 0.00 0.00 0.00

```

Extraits de code – Evaluation du réseau

Pour chaque neurone: somme pondérée des entrées (ici, toujours deux entrées non nulles: iTh et idTh)

```

void EvaluationDuReseau(tlIndex ITh, tlIndex IdTh)
{int i;
 for(i=1;i<=NNeurones;i++)
  ReseauNeurone[i-1].Sortie =
  ReseauNeurone[i-1].Poids[ITh-1] + ReseauNeurone[i-1].Poids[NEntreesParVar+IdTh-1];
}

void RechercheDuNeuroneLePlusActif(int &IMax)
{ int i;
 float s=0;
 IMax=0;
 for(i=1;i<=NNeurones;i++)
  if(s<ReseauNeurone[i-1].Sortie)
   {s=ReseauNeurone[i-1].Sortie;
   IMax=i; } // end for et if
}

```


Extraits de code – Processus principal (Version C++ - HEIG-VD.iAi.LaRA)

```
void CommandeNeuronale(float T, float dT, float &Co)
{
    if(T > 180) T=T-360;
    if(T < -180) T=T+360;
    int ITh, IdTh, IMax;
    Single2Index( T, ThMin, ThMax, NEntreesParVar, ITh);
    Single2Index( dT, dThMin, dThMax, NEntreesParVar, IdTh);
    EvaluationDuReseau (ITh, IdTh);
    RechercheDuNeuroneLePlusActif(IMax);
    Form1->Edit1->Text=IntToStr(IMax) ;
    if(IMax>0)
    {
        ReConversionFormat(IMax, Co );
        IMaxOld=IMax;
    }
    else
    {
        IMax=IMaxOld;
    }
}

...
```

Extraits de code – Processus principal (Version Pascal - HEIG-VD.iAi.LaRA)

```
Procedure CommandeNeuronale(T, dT: single; var Co: single );
var
    ITh, IdTH, iMax: integer;
begin
    Single2Index( T, ThMin, ThMax, NEntreesParVar, ITh);
    Single2Index( dT, dThMin, dThMax, NEntreesParVar, IdTh);
    EvaluationDuReseau (ITh, IdTh);
    RechercheDuNeuroneLePlusActif(iMax);
    if iMax>0 then ...
        ReConversionFormat( iMax, Co ); ...
    else
        iMax:=iMaxOld;
end;
```

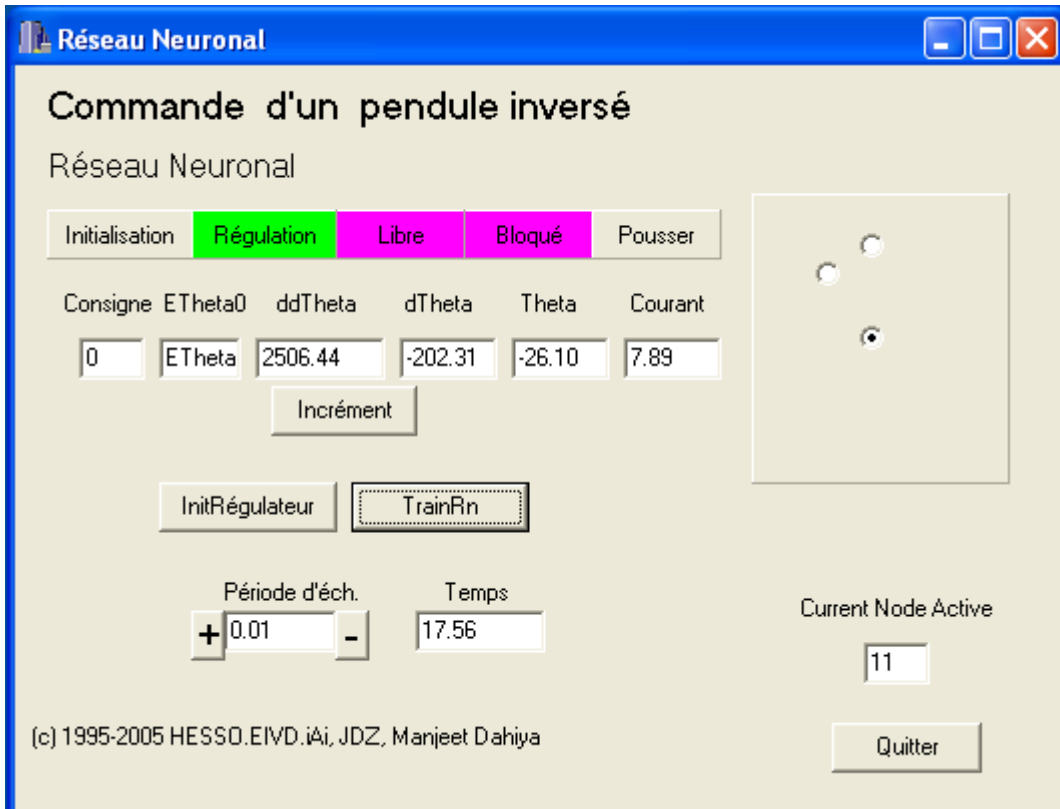


Fig.6.2.12 Régulation d'un pendule inverse par réseau neuronal (doc. Hesso-HEIG-VD-iAi)

Extraits de code – Simulation d'un balancier

```
void UpdateBalancier(/*float Courant,
    float &Theta, float &dTheta, float &ddTheta*/)
{
    ddTheta=360/Lm*g*sin(Theta/180*pi)+300*Courant;
    dTheta=0.99*dTheta+ddTheta*dt;
    Theta=Theta+dTheta*dt;
    t=t+dt;
    return;
}
```

6.2.6. Conclusion

Le document présente une brève introduction à la commande par réseau neuronale.

De façon plus développée, il décrit les modules logiciels créés à l'HEIG-VD pour un cas classique en régulation: la commande d'un pendule maintenu en position verticale.

Pour plus de détails en ce qui concerne la théorie, on se reportera aux ouvrages spécialisés.

Pour plus d'information en ce qui concerne l'application concrète, on peut consulter le code C++ (dans nos archives, également, enPascal) disponible au laboratoire, et/ou essayer de façon interactive les modules exécutables.

6.2.Références

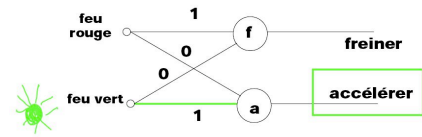
- [1] Liardon , "Etude et programmation de réseaux neuronaux", rapp. de mini-projet, LaRA, HEIG-VD, mars 1997, pp. 15 env.

- [2] SOMA - SIMULATEURS DE RESEAUX DE NEURONES, Cours et polycopié COMETT, EPFL/LAMI, 1993?
- [3] P. Valsecchi, "Prévision de séries temporelles par réseaux de neurones artificiels", rapp. de diplôme, HEIG-VD, 1995.
- [4] J.-D. Dessimoz, "Intelligence artificielle et génération de plans d'action", HEIG-VD, 1996, pp. 90 env.
- [5] François Blayo, "Introduction aux réseaux neuronaux", EPFL/LAMI, mars 1991, pp.111
- [6] S.Y.Kung, "Digital neural networks"

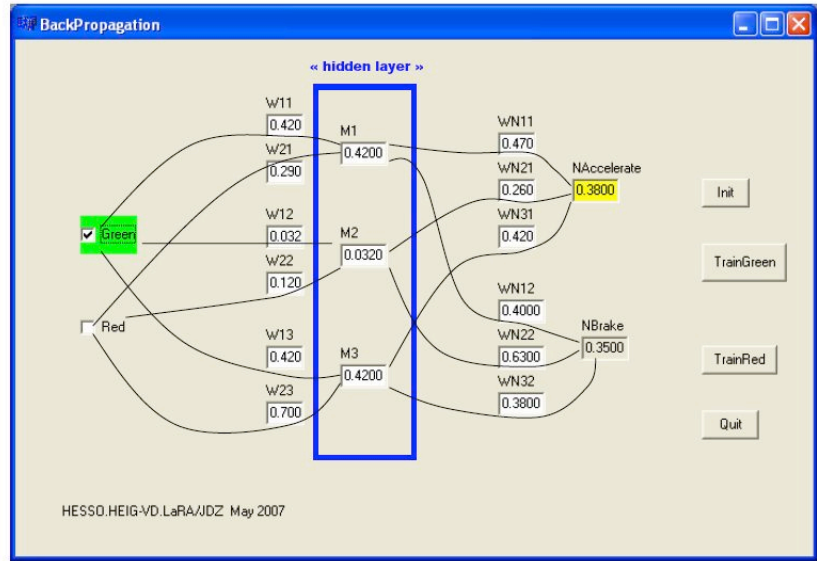
6.3 Autres réseaux de neurones

Le réseau formé de neurones du type de Hopfield et utilisant la loi d'apprentissage de Hebb est le plus fondamental, mais il y existe aussi d'autres types de réseaux neuronaux, spécialisés, itératifs, avec boucles internes:

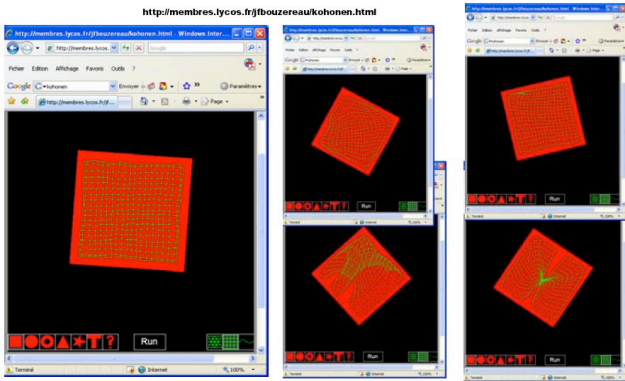
- Réseau $\hat{}$ rétropropagation: nous avons ici plusieurs couches. Les poids sont d'abord tous définis au hasard, puis, avec une sorte de généralisation de la loi de Hebb, sur la base de chaque exemple fourni, les coefficients sont renforcés en cas de succès ou au contraire diminués, en cas d'échec.
- Réseau de Kohonen: les liaisons sont itérativement adaptées de façon $\hat{}$ ce que des liaisons voisines en entrée activent également des neurones voisins en sortie
- Réseau $\hat{}$ compétition-coopération: des liaisons transversales entre neurone renforcent ou au contraire inhibent les neurones voisins
- Réseau de Héroult-Jutten: ce réseau particulier permet le découplage de signaux a priori mélangés (diaphonie)



a

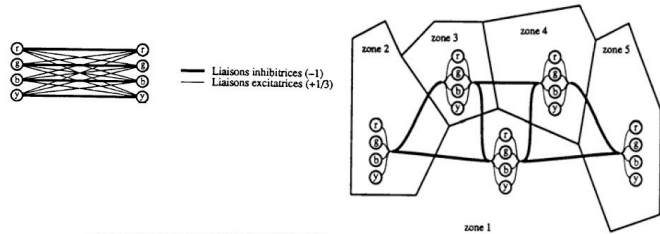


b



c

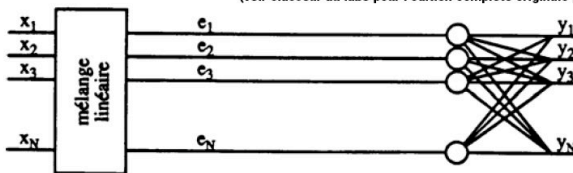
Carte colorée par relaxation:



d

SOMA: SIMULATEURS DE RÉSEAUX DE NEURONES
cours COMETT-Neural
Résumé et mis en page par EINEV/JDZ
(voir classeur au labo pour l'édition complète originale par LAMIE/EPL)

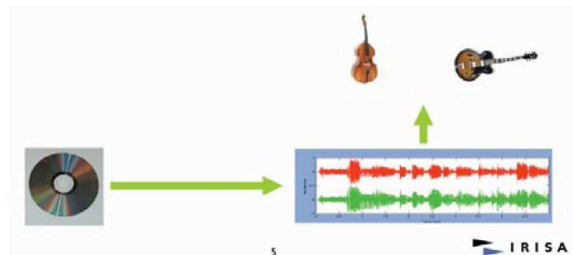
Contexte: diaphonie



Signaux, mélange et séparation:



e



f

Fig.6.3.1 Exemples de réseaux neuronaux du laboratoire avec solutions élémentaires de base, (a), à rétropropagation (b), ou visant des types d'applications spéciales, notamment la réorganisation topologique (Kohonen, c), le coloriage de cartes (d), ou encore la séparation de sources (Hérault-Jutten, f).

7 ALGORITHMES GENETIQUES

Les algorithmes constituent une autre tentative bioinspirée pour créer de nouvelles solutions. Nous les présentons d'abord de façon théorique, puis illustrant ces concepts sur notre processus de type industriel, fil rouge, la commande de pendule inverse.

7.1 Définitions générales pour les algorithmes génétiques

Ci-dessous, une description faite essentiellement par M. Faeh, dans le cadre d'un cours à option (HEIG-VD/JDZ) donné en 1997.

7.1.1 Définitions:

En tant que méthode de recherche, l'approche génétique consiste à utiliser un ensemble de solutions potentielles auxquelles on attribue des valeurs. En considérant une solution comme un individu et en assimilant les caractéristiques d'une solution aux composants d'un génome, le GA fait évoluer une population de solutions en y propageant les caractéristiques des individus les plus valables qui, suivant la métaphore évolutionniste, sont les plus adaptés. Les individus ainsi soumis à une sélection et une évolution vont s'adapter et trouver une niche écologique qui correspond à un maximum dans l'espace de recherche. Les maximums locaux peuvent en outre être surmontés par le système car les individus qui la composent sont, tout comme dans la nature, soumis à des mutations aléatoires engendrant de nouvelles caractéristiques.

Avant d'entrer plus avant dans la description d'un GA il est utile d'introduire le vocabulaire que nous allons utiliser:

- individu, organisme: un organisme biologique est plus qu'un génome, c'est aussi une forme, un phénotype qui est le produit de l'activité des gènes. Dans le cadre d'un GA traditionnel, l'individu est réduit à son génome, à un ensemble de caractéristiques élémentaires.
- chromosome: un GA classique traite des individus constitués d'une seule séquence de gènes, un chromosome, alors que les organismes biologiques peuvent en avoir plusieurs. L'approche informatique idéalise traditionnellement le chromosome en le réduisant à un vecteur de gènes alors que la molécule d'ADN possède une configuration dans l'espace.
- gène: caractéristique d'une solution au problème que le GA traite. Il faut noter que lorsque qu'on parle d'organisme vivant, un gène est souvent plus qu'un objet atomique. Un gène peut être un complexe de séquences génétiques qui coopèrent pour assurer une fonction.
- locus: signifie lieu en latin Le locus est la position du gène dans le chromosome. Ce terme vaut pour le GA aussi bien que pour les systèmes biologiques.
- allèle: (du grec allos, autre) symbole attaché à un gène. Alors que l'alphabet du code génétique naturel est composé des quatre lettres ATCG nous pouvons choisir n'importe quel alphabet pour un GA. Toutefois, un alphabet binaire est couramment utilisé.
- population: un groupe d'organismes artificiels ou naturels.

7.1.2 Opérateurs génétiques

Un algorithme génétique élémentaire consiste en une population d'entités possédant des caractéristiques que l'on soumet à une sélection puis aux opérateurs génétiques suivants:

- la reproduction: qui consiste à dupliquer un individu dans la population, donc à copier ses caractéristiques, c'est-à-dire ses gènes.
- l'hybridation: (ou crossover) qui recombine deux individus parents en produisant deux nouveaux individus qui héritent des caractéristiques parentales.
- la mutation: qui est une altération aléatoire des gènes d'un individu.

Précisons la signification de ces termes:

La mutation

La mutation met en jeu un seul chromosome (individu de la population du GA) et provoque une modification de certains de ses gènes. Le gène modifié peut provoquer un accroissement ou un affaiblissement de la valeur de la solution que représente l'individu. Ce mécanisme est similaire

à celui que l'on observe dans la nature si l'on omet l'existence de gènes récessifs et dominants qui peuvent en limiter l'effet. Tout comme dans la nature, une mutation peut aussi procurer à l'individu d'un GA un avantage en terme de sélection et son caractère aléatoire peut permettre au GA d'échapper à un maximum local de l'espace de recherche.

Le crossover

Le crossover (enjambement) utilisé par les GAs est la transposition informatique du mécanisme qui permet, dans la nature, la production de chromosomes qui héritent partiellement des caractéristiques des parents. Principale propriété du crossover: l'information génétique globale est préservée, les gènes étant simplement transférés d'un individu à l'autre. C'est le crossover qui permet l'exploitation, par combinaison, des "briques de bases" sélectionnées au cours des générations passées.

L'hybridation et le codage des caractéristiques

Il est aisé de constater comment un mauvais codage peut diminuer considérablement les performances d'un algorithme génétique en comparant deux des codages possibles d'une caractéristique pouvant prendre quatre valeurs différentes:

un codage symbolique, où l'on décide que la caractéristique est représentée par les quatre symboles: A, B, C, D;

un codage binaire, dans lequel cette même caractéristique est représentée par quatre chaînes de: 00, 01, 10, 11.

Dans le cas d'individus ne possédant qu'une seule caractéristique codée symboliquement, l'hybridation est impossible car il n'y a aucun moyen immédiat de combiner deux symboles. Le seul opérateur applicable est la mutation qui substitue aléatoirement un symbole à un autre. En revanche, le codage binaire permet de recombinaison deux caractéristiques par hybridation. Les caractéristiques d'un individu (assimilables à des gènes contenus dans un chromosome) doivent donc être codées de façon à supporter l'application efficace des opérateurs génétiques

7.1.3 Cycle d'activation

Le cycle d'activation d'un algorithme génétique peut se décomposer comme suit:

1. évaluation des individus d'après leurs caractéristiques; 2. reproduction des individus par sélection aléatoire pondérée en fonction de la valeur obtenue en l 3. hybridation entre individus de la nouvelle génération; 4. application de mutations aléatoires.

Il est toutefois possible de combiner les étapes 2, 3 et 4 en une seule. Le mécanisme de reproduction est alors, comme dans la nature, responsable de l'hybridation (par enjambement de chromosome) et de la mutation (par erreur de copie).

7.1.4 Le contrôle et la mise en oeuvre d'un GA

Un GA est traditionnellement contrôlé par 4 paramètres:

- la fonction d'évaluation qui détermine la probabilité de sélection et de reproduction d'un individu.
- la taille de la population: qui détermine le nombre de gènes dont le GA dispose pour produire de bonnes solutions. Une taille de population trop faible peut empêcher le GA de traiter certains gènes ou peut encore limiter sa recherche si la fonction à optimiser (fitness landscape) possède de nombreux maxima locaux.
- la probabilité de mutation que subit chaque allèle lors de la reproduction.
- la probabilité de crossover qui détermine la fréquence à laquelle les hybridations entre individus vont avoir lieu.

Ces deux derniers paramètres sont corrélés. Alors que la mutation permet l'apparition de nouveaux gènes par la modification d'un allèle, le crossover diffuse les gènes existants lors du renouvellement de la population. Cette balance entre l'exploration par mutation et l'exploitation par crossover doit être soigneusement respectée car un taux de mutation trop élevé entraîne la

destruction de gènes avant qu'ils n'aient eu la chance d'être assemblés par crossover afin de former des structures valables. De l'autre côté, si le taux de crossover est excessif, la population sera uniformisée trop rapidement et la population convergera alors prématurément vers un type d'individu probablement sous-optimal. On ne parle alors plus de la survivance du plus apte, mais de la survivance du plus médiocre.

7.1.5 Application d'un algorithme génétique à l'estimation de mouvement d'objets

L'estimation du mouvement d'un ou plusieurs objets dans une séquence permet de résoudre de façon élégante les problèmes posés par certaines applications, dont: les systèmes de surveillance vidéo les studios d'enregistrement vidéo automatisé

- la compression du signal vidéo (MPEG-4)

Contrôle et mis en oeuvre:

Notre problème consiste donc à retrouver la position d'un objet (individu) donné dans l'image suivante (population).

Définition: . les individus Comme nous l'avons vu plus haut, il va s'agir de traiter des individus. Dans notre cas ces derniers représentent une fraction de l'image à analyser (bloc). Lors du codage binaire de ces blocs, il faut essayer de ne tenir compte que des caractéristiques portant de l'information.

- La population L'image dans laquelle le mouvement doit être exprimé, est divisée en blocs de taille identique à l'individu. La technique de codage pour ces blocs doit être identique aussi.

Cycle d'activation de l'AG pour une estimation de mouvement

1. Quelques individus sont sélectionnés de façon aléatoires dans la population. 2. Seule les individus comportant une certaine ressemblance avec l'individu recherché survivent, les autres sont tués. 3. hybridation entre individus de la nouvelle génération. 4. application de mutations aléatoires.

Le cycle 2-3-4 est répété jusqu'à l'obtention d'un individu de la population équivalent à celui recherché. La position correspond environ à celle des parents. Tout le problème consiste à trouver le bon paramétrage pour que cet algorithme soit applicable au problème posé. Les résultats obtenus par ce genre d'algorithme sont très prometteurs, par contre des difficultés majeures sont apparues lors de l'intégration sur circuit VLSI (MPEG-4).

7.1.6 Conclusion:

Les algorithmes génétiques appliqués à l'estimation de mouvement permettent de résoudre ce problème avec très peu de calcul à effectuer en comparaison de l'approche classique. La difficulté principal consiste à adapter le modèle à une utilisation spécifique. La mise en oeuvre peut paraître un peu floue, mais il faut préciser que je n'ai pas pu avoir plus de précision au sujet de la réalisation pratique ni au sujet de l'implémentation.

7.2 Réalisation HEIG-VD - Algorithme génétique pour la régulation d'un pendule inversé

L'exemple ci-dessous a été traité par Stéphane Emmenegger en travail de semestre/diplôme 2000 [Emm]. Didacticiel et présentation Powerpoint sont téléchargeables sur le web (<http://HEIG-VD.ch/iai>, projet robotique).

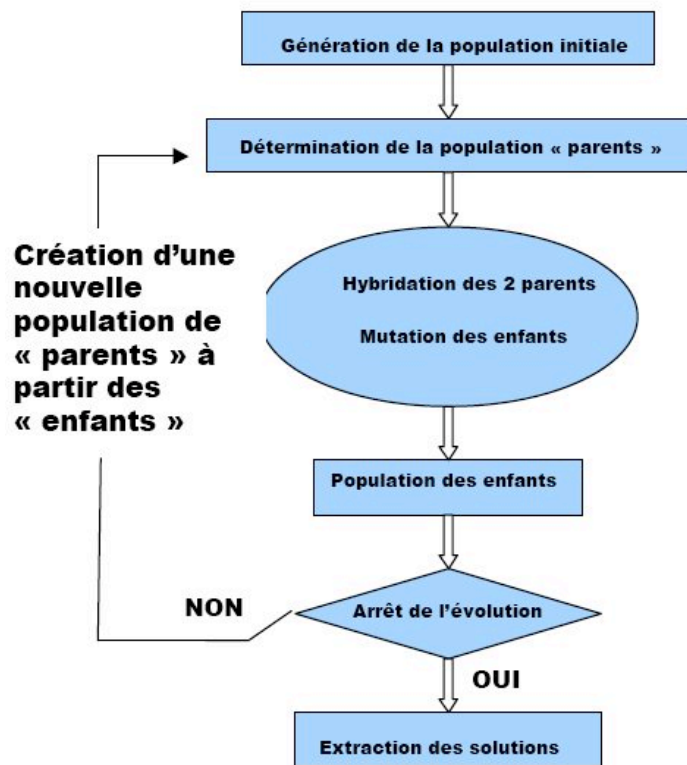


Fig. 7.2.1 Réglage d'un pendule à l'aide d'algorithmes génétiques

Il s'agit d'une famille d'algorithmes basés sur l'évolution de l'espèce.

L'évolution d'une espèce est simplement une suite successive d'améliorations afin qu'elle soit le mieux adaptée au milieu dans lequel elle évolue.

Fig. 7.2.2 Organigramme de l'algorithme



Les algorithmes génétiques, aussi appelés « de sélection naturelle », travaillent sur des chaînes de caractères particulières (chromosomes) généralement définies de manière aléatoire.

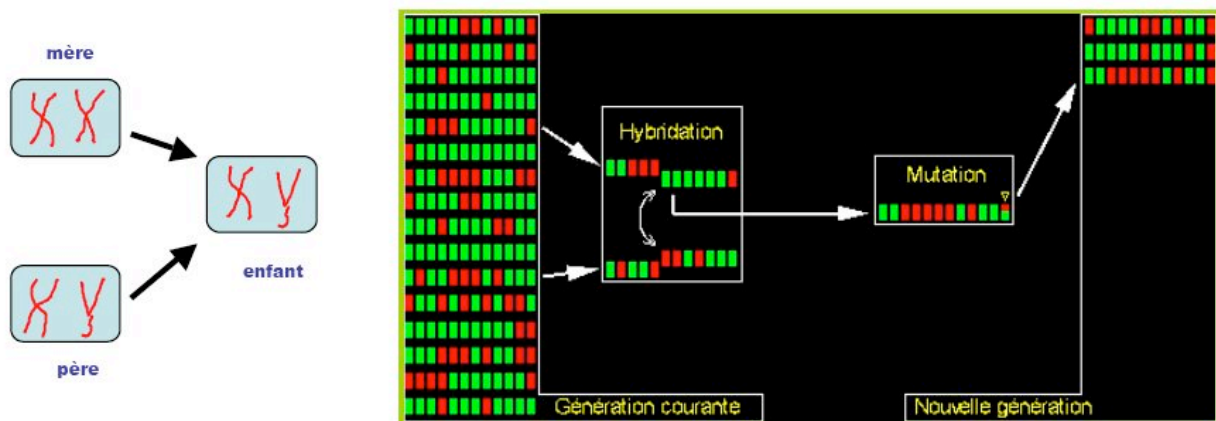


Fig. 7.2.3 Illustration des étapes de génération par hybridation et mutation éventuelle.

Les individus vont s'évaluer, se sélectionner, se reproduire et subir des mutations

Le logiciel RaGE développé permet de simuler graphiquement un pendule inversé en mode libre ou régulé.

Le type de régulation employé est une régulation proportionnelle-dérivée

Le dimensionnement du régulateur peut se faire soit manuellement, soit, et c'est la partie la plus originale, par l'intermédiaire d'un algorithme génétique.

Phases	Actions	Paramètres de départ
Initialisation	Génération d'une population de régulateurs de manière aléatoire.	Population maximum
Hybridation	Composition par paires des chromosomes de la population pour générer des enfants.	Nombre de générations
Mutation	Modification aléatoire d'un des éléments du chromosome (phénomène « rare »).	Probabilité de mutation
Sélection	Les 75% les plus performants sont conservés.	Critère de sélection

Fig. 7.2.4 Les phases principales d'un algorithme génétique sont classiquement au nombre de 4.



Fig. 7.2.5 Ecran de commande principal. Dans le cas particulier, les régulateurs sont sélectionnés sur la base de leur proximité au "gain optimal", pour simplicité de la mise en œuvre de l'AG. Mais on pourrait aussi prendre le "temps de stabilisation" si ce gain optimal était a priori inconnu



Fig. 7.2.6 Paramètres et résultats partiels de l’algorithme génétique implémenté. Dans la version C++, c’est le temps de montée du régulateur, en mode simulé, qui sert de critère de sélection ; l’idéal serait zéro.

En résumé, les algorithmes génétiques ont surtout la propriété de générer de façon aléatoire des solutions, en s’inspirant des mécanismes du vivant. Fondamentalement, le hasard peut générer n’importe quelle solution, donc aussi la meilleure. Reste à réaliser un générateur (vraiment) aléatoire, et à prendre le temps d’explorer un nombre généralement très grand, voire prohibitif (!) de solutions potentielles.

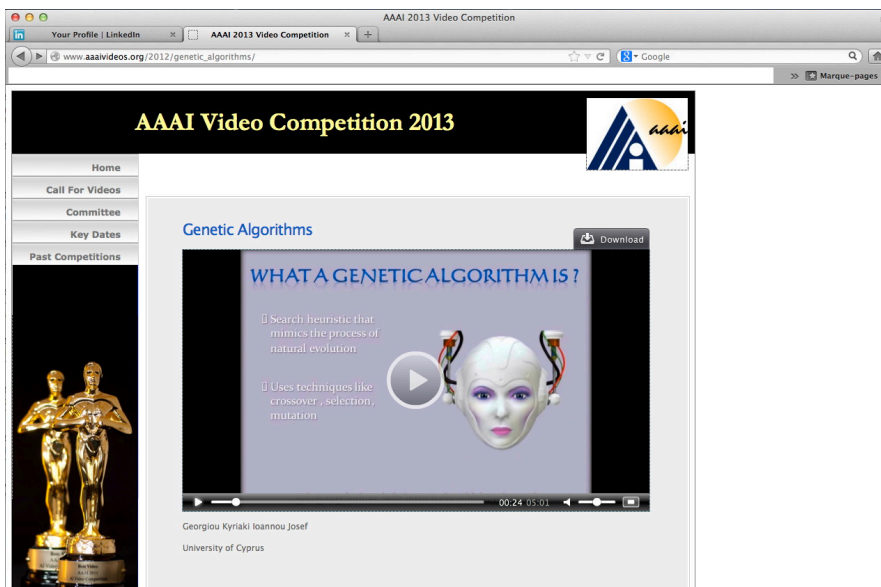


Fig. 7.2.7 Description récente, sans changement, de l’approche à AG décrite plus haut.

8. ROBOTS COOPÉRATIFS ET HUMANOÏDES

Considérons le passage entre les animats sans modélisation vus plus haut (cf. p. 32) et un objectif ambitieux concernant la coopération de robots intelligents avec l'humain.

Pour atteindre ce type de performances, il faut d'une part s'appuyer sur des robots intelligents, ce qui implique forcément une modélisation dynamique d'eux-mêmes et de leur environnement, comme dans l'exemple donné plus haut d'un labyrinthe coloré.

Il faut aussi avoir une base solide en termes de robots mobiles autonomes robustes, tels que typiquement développés dans le contexte de la compétition européenne Eurobot. Pour ce dernier contexte, le LaRA a développé de multiples plateformes et un environnement original de commande et de programmation temps-réel, pour le monde réel, « Piaget ».

Voyons d'abord ces robots robustes autonomes, puis, dans une deuxième section nous verrons les robots coopératifs qui étendent encore les possibilités à la communication avec l'humain.

8.1 Robots mobiles autonomes robustes – Exemple Eurobot

Considérons l'exemple de Lomu, développé pour Eurobot 2005, et qui est encore parfaitement représentatif aujourd'hui (2014). Le but était de gagner un maximum de points en 90 secondes, en allant ramasser des « noix de coco » et en les lançant dans le camp adverse à la façon du rugby. Un robot auxiliaire, éventuellement défenseur était aussi possible.

Les figures qui suivent illustrent les éléments principaux, matériels méthodologiques et logiciels des systèmes réalisés.

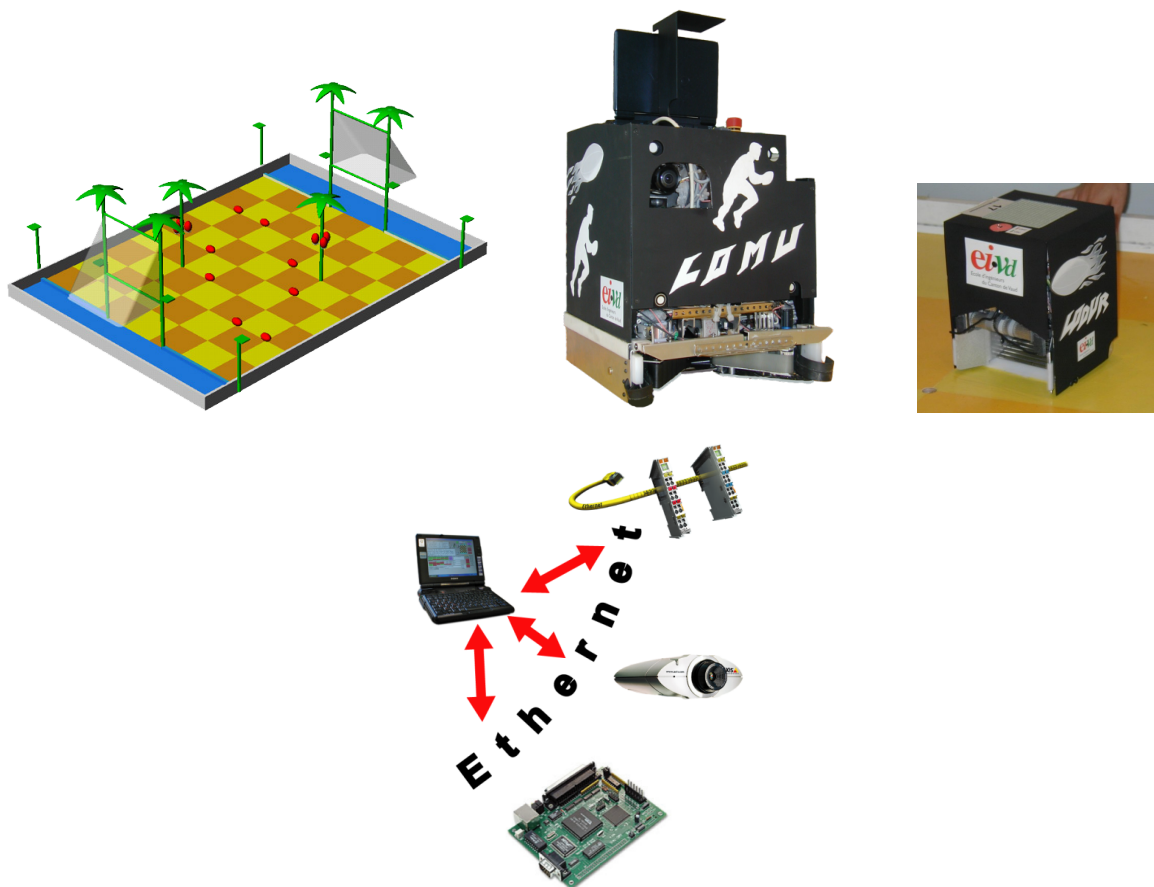


Fig. 8.1.a Table de jeu, robots grand et petit, et composants majeurs

L'architecture du robot est la suivante:

1. Un hub qui interconnecte physiquement les différents composants
2. Un ordi compact style notebook, qui est notre infrastructure pour centre de décision
3. Une commande de mouvement spécialisée, avec lois de mouvement pour trajectoires
4. Une caméra vidéo
5. Un automate programmable pour la gestion des entrées sorties rapides et robustes

Sur le plan méthodologique, la gestion des positions et des orientations jouent un grand rôle (cf. Fig. 8.1b).

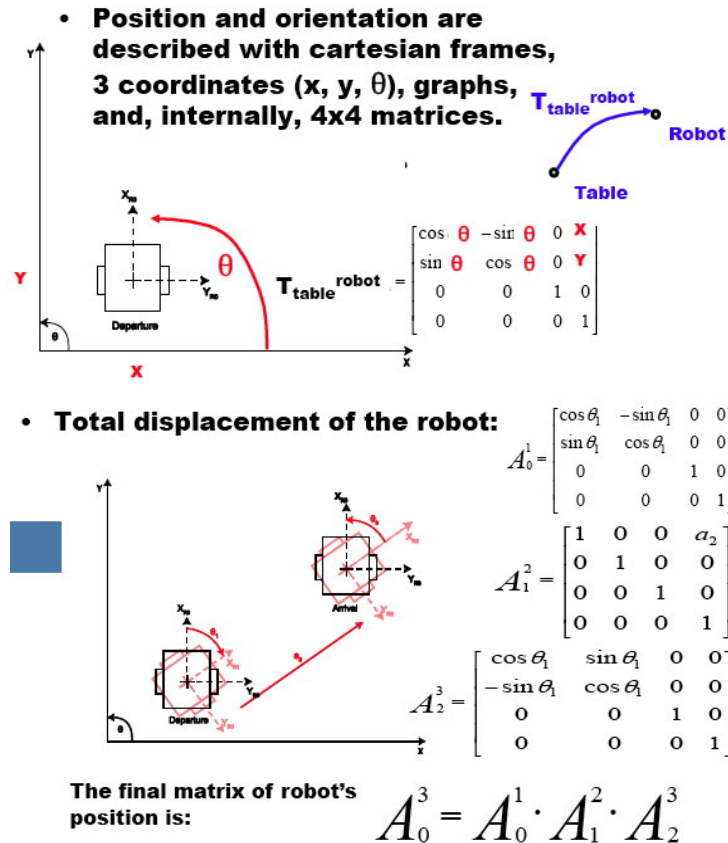


Fig. 8.1.b Positions, coordonnées spatiales, matrices et graphe des transformations

La programmation est multi-tâches et, avec Piaget, intègre des instructions de type VAL, sans doute le meilleur langage pour robots industriels.

```

// PIAGET et quelques instructions de style VAL (V
+)
location Position1 ; // exemple : position No 1
int NSIDemarrage=1 ; // exemple : entrée No 1
int NSOCanon=2 ; // exemple : sortie No 2
...
// Début du match
50: WaitAGN(NSIDemarrage); //attente d'une entrée
    « vraie »
    break; case
51 : GoState(1001); // break; case
...
// Tir des balles de ping pong
500: SignalOutAGN(NSOCanon, true); //commande d'une
    // sortie
    break; case
...
// Affichage
602: TypeStringAGN(" US Gauche: "); break; case
603: TypeIntegerAGN(DistanceCapteurUSGauche());
    break; case
604: SleepAGN(1); // attente d'une seconde
    break; case
605: ; // arrêt de la tâche
...
721: SetAGN(Position1,Trans(100,100,-90));
    // on définit une position en x,y cm,
    // avec orientation de phi degrés
    break; case
722: MoveAGN(Position1); // on s'y déplace
...
1001: if(SignalIn(NSIBARRIEREEntree)) //test d'une
    entrée
    GoState(1130); // Palet ou...

```

Fig. 8.1c. Instructions Piaget, notamment pour la gestion des entrées-sorties, la spécification des positions et orientations, ainsi que les ordres de mouvement

Beaucoup d'information s'acquiert par la perception visuelle pour permettre l'adaptation à l'environnement, évitement d'obstacles, navigation, prise, transport et manipulation d'objets.

Image processing:

- extrait le vert-brun cocotier
- intègre verticalement
- extrait la zone maximale (vert foncé)
- intègre horizontalement
- filtre avec médian
- localise transition verticale

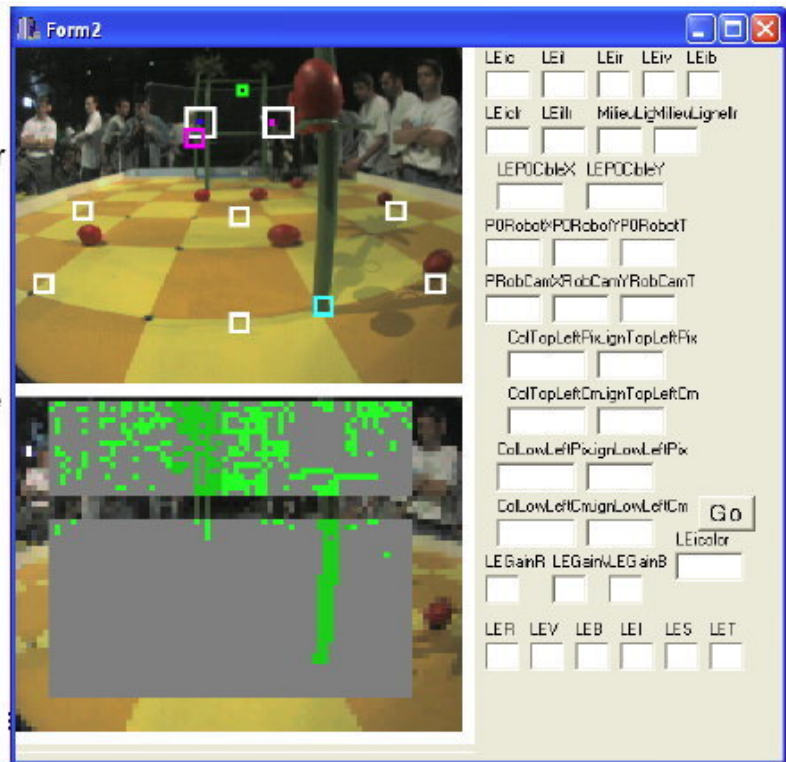


Image processing:

- Equilibre des blancs
 - GainsRVB
- Extrait neuf couleurs:
 - Saturation faible
 - Noir, gris ou blanc
 - Saturation forte
 - Rouge, vert, bleu
 - Jaune, cyan, magenta

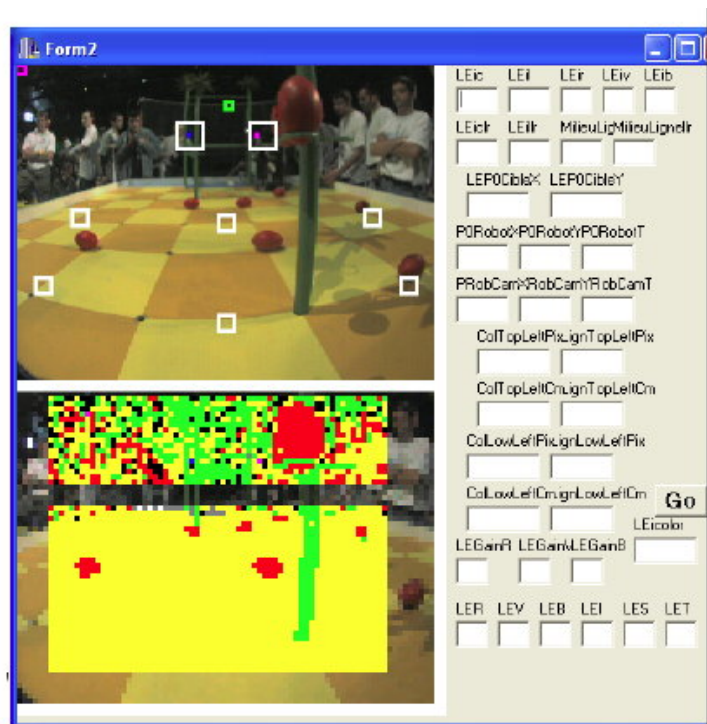


Fig. 8.1d. Exemples de vision pour localisation des cocotiers, des noix de cocos, ainsi que des balises proches des buts de rugby.

8.2 Robots coopératifs à capacités cognitives développées – Exemple Robocup@Home

L'initiative la plus significative à la frontière des sciences cognitives, de l'IA et de la robotique est celle qui a démarré à la fin des années 1990, visant l'invention de robots footballeurs et de systèmes similaires, notamment pour des applications domestiques (cf. Fig. 8.2a).

Des informations à ce sujet, en plus du cours de robotique (yc. Piaget et capteurs divers, tels Kinect), peuvent se trouver aux adresses suivantes d'internet : www.robocup.org pour l'initiative en général, et <http://rahe.populus.ch> pour les contributions du LaRA dans ce contexte.



Fig. 8.2a. Robocup : Match de foot en Chine en 2008, Japon-Allemagne (Osaka-NimRo, a,b), plateforme standard NAO (c), Robot RH4-Y de la HEIG-VD à la compétition de Graz, dans la compétition pour assistance domestique (d), et Avatar (film de Cameron 2009, e)

Voici quelques illustrations relatives à des composantes importantes de notre projet et groupe de robots coopératifs développés pour l'assistance en milieu domestique.

Un élément critique est l'environnement de commande interactive et de programmation pour le monde réel « Piaget ». D'abord inventé pour nos robots engagés dans les compétitions Eurobot, Piaget a depuis ajouté bien des fonctions pour la coopération naturelle avec l'humain et la commande de robots industriels également.,

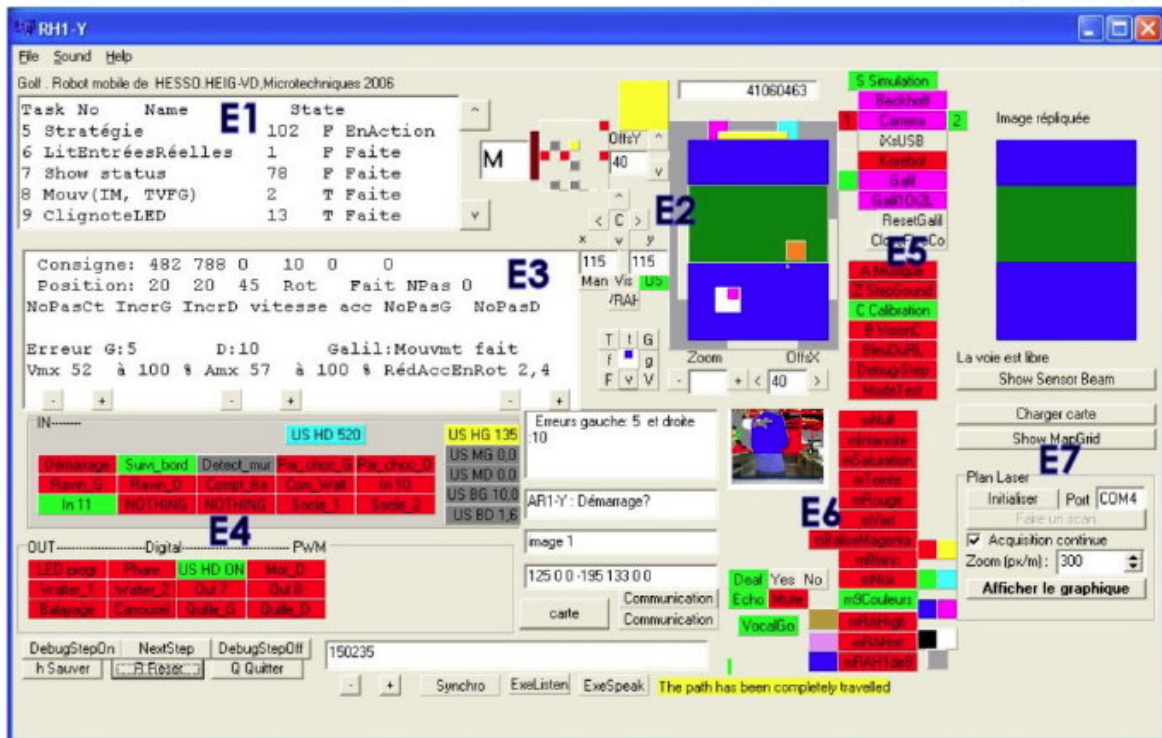


Fig. 8.2b. Panneau principal de commande et de programmation de l'environnement Piaget. La version présentée ici ne contient pas encore de multiples autres applications et fiches auxiliaires ajoutées dans les versions suivantes.

Suivant la phase de travail et l'application considérée, diverse fiches auxiliaires deviennent visibles.

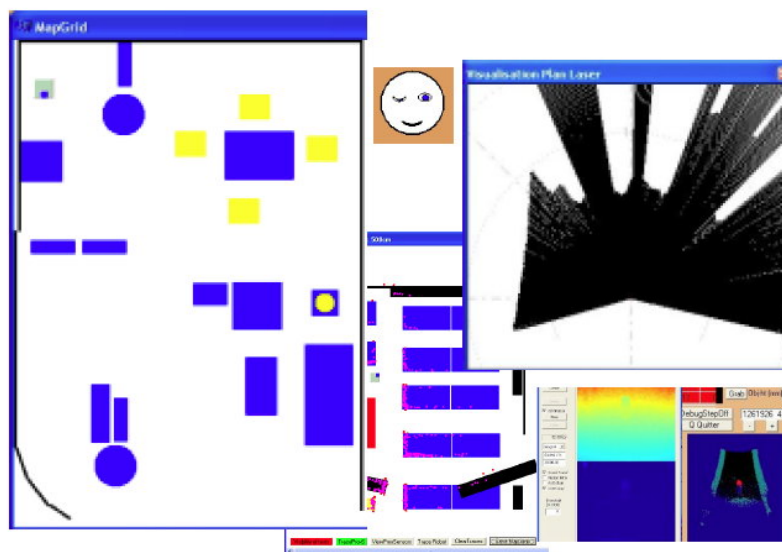


Fig. 8.2c. Fiches de l'environnement Piaget montrant des cartes topologiques, l'expression d'émotions antropomorphiques, ainsi que des .mesures de distances planes ou bidimensionnelles à temps de vol.

La figure suivante illustre deux applications emblématiques. La première illustre un groupe (RG-Y) de robots apportant, à la demande, des boissons et des snacks à un humain. Il s'y trouve notamment un humanoïde Nono-Y (de type NAO, réalisé et commercialisé par Aldebaran Robotics) jouant le rôle de médiateur entre l'humain et les autres robot/machines qui font effectivement le travail (RH-Y, OP-Y).



Fig. 8.2d. Deux applications : « Open Challenge », durant la compétition Robocup@Home à Singapour, en 2010 (en haut); .et « TeleGrab », démontré au LaRA en 2013. Vidéos disponibles.

La communication entre NAO et ordinateur externe, et plus spécifiquement Piaget, a nécessité des développements originaux illustrés en Fig. 8.2e.

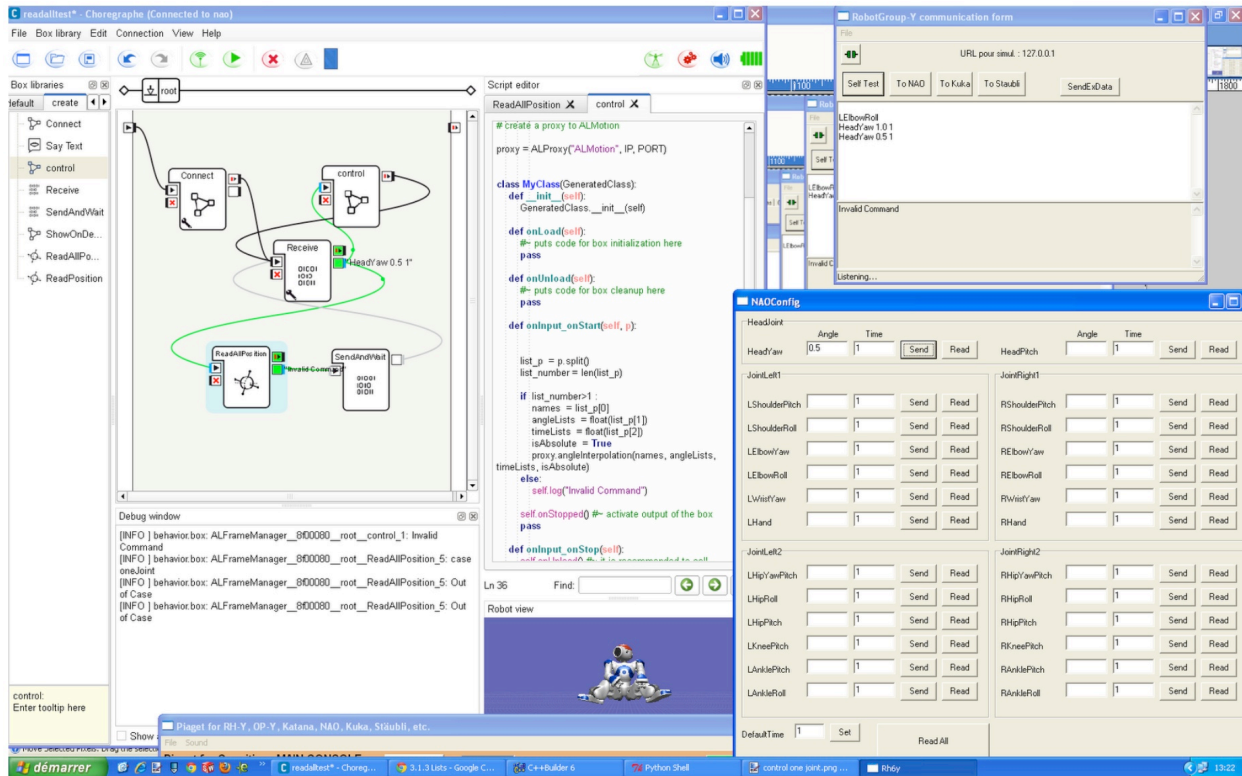


Fig. 8.2e. Programmation et commande de NAO en contexte Piaget, permettant la gestion du groupe de robots avec leur divers moyens auxiliaires (vision, reconnaissance vocale, perception des distances, etc.) : code avec blocs spéciaux de type chorégraphie pour communication TCP-IP (à gauche), code Python d'implémentation (centre), communication par socket sur ordi (en haut à droite) et fiche de surveillance et de commande temps-réel pour toutes les articulations de NAO en Piaget.

La deuxième application, « TeleGrab », permet à un humain de bénéficier de l'aide de robots pour prendre à distance des objets et de les lui apporter. Trois composantes principales se distinguent : un mode global autonome (par exemple la navigation : « Va à la cuisine ») ; un mode détaillé de type « esclave » (par exemple une légère rotation à droite, ou l'ouverture progressive de la « main »), qui peut aussi servir de base pour de nouveaux apprentissages ; et enfin un mode adaptatif intermédiaire (par exemple saisie automatique d'un objet se trouvant à moins d'un mètre, ou encore le suivi d'un guide).

Les applications ci-dessus exploitent certaines composantes très évoluées : reconnaissance visuelle et de gestes, ou encore, comme illustré dans la figure suivante, reconnaissance vocale, conscience de soi et localisation dans l'espace, notion de groupe à culture partagée, programmation graphique et en réseau, ou encore médiation humain-robots.



Robot Group (RG-Y) and wifi-based internal cooperation; Nao as a mediator ; Piaget integration; RH with new power drive; tablet PC in the group.

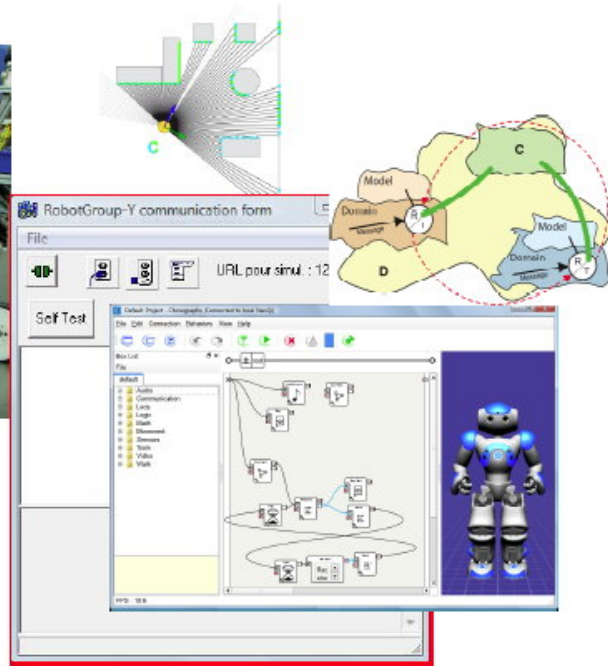
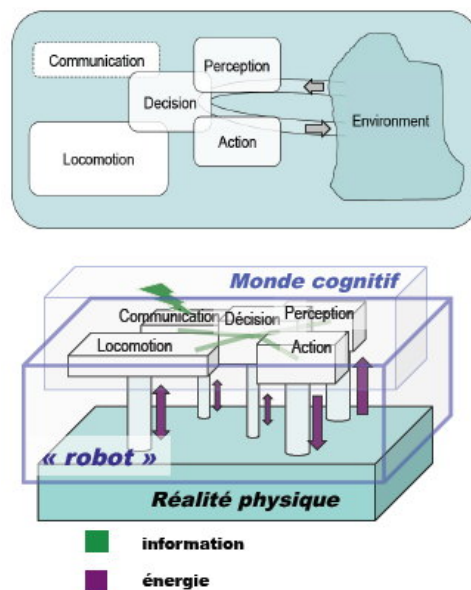


Fig. 8.2f. Le groupe de robots RG-Y comprend OP-Y, Nono-Y « notre » NAO, et RH-Y (photo de gauche). Par corrélation 3D (x,y, phi) de cartes 2D (calculées sur la base de données polaires saisies par « ranger » plan, le robot peut se situer dans l'espace et recalibrer sa pose en revenant dans des régions connues (centre haut). Pour réaliser un groupe, les agents cognitifs individuels communiquent et partagent une culture commune (en haut à droite). NAO se programme notamment de façon graphique (en bas à droite) mais se programme et se gère aussi en temps réel via wifi, Ethernet, TCP-IP, via l'environnement Piaget (centre, en bas).

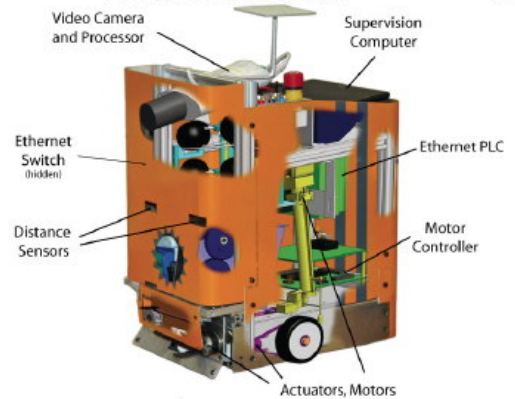
Les robots évolués qui sont rapportés ici jouent un rôle révolutionnaire en termes « d'embodiment », c'est-à-dire d'intégration entre mondes cognitif et monde physique, réel, avec une grande variété de circonstances, vu leur mobilité (cf. Fig.8.2f). Dans le passé, seul l'humain pouvait vrai**Aucune entrée d'index n'a été trouvée.**ment le faire.

Fig. 8.2g. Les robots permettent de relier le monde cognitif au monde réel, physique. L'implantation d'un agent cognitif dans le monde physique, cela correspond au concept anglais d'«embodiment».



```

11: SleepAGN(0.05);
12: if (!SignalIn(NSISStart))
    GoState(6);
    else
    GoState(20);
20: DemarrerMatchAGN(); // start 90 s tin
21: SignalOutAGN(NSOAspirateur, true); // start n
22: SignalOutAGN(NSORouleauIN, true); // start m
23 : ApproAGN(HoleNb1, 15);
    
```



Les robots humanoïdes ont par définition la forme grossièrement humaine (notamment deux bras, et deux jambes) ; ils ont actuellement les défauts d'être coûteux et peu stables. Mais à terme ils auront l'avantage de pouvoir remplacer l'humain pour toutes les tâches où une cinématique et des actions similaires sur l'environnement sont nécessaires.

9. AUTRES EXEMPLES IMPLIQUANT COGNITION ET COGNITIQUE AU LARA

Le chapitre a déjà présenté de nombreux exemples de ce qui se fait en I.A. Il se termine ici en présentant encore trois exemples concrets du LaRA.

Exemple


heig-vd

Haute Ecole d'Ingénieurs et de Gestion du Canton de Vaud

➤ **Boîtes de films et productique**


Problem specs (cognitive terms)

input information: 60 [bit]
 output inf.: 3e4 [bit]
 complexity: 3e54 [bit]
 knowledge: 202 [lin]
 expertise: 3.3 [lin/s]



problem space

Classes of solutions



coercive adaptive cognitive



Adaptive system
(robot with vision)



video camera



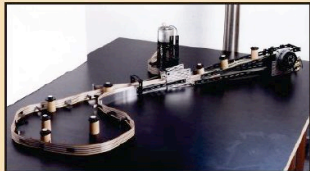
gripper






playground & access zones




image analysis



Coercive sytem
(no sensor, no data processor)

Example of cognitive approach



What is a film can ?




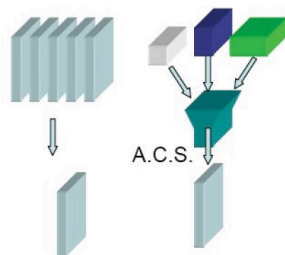
Fig. 9.1. Pour la compétition Per'Ac, il s'agissait d'empiler des boîtes de films en 1 minute. Les points correspondaient au carré de la hauteur des piles faites. Le problème a été résolu de deux façons : par robot industriel et vision par ordinateur ; et par un système sans capteur ni processeur numérique. On constate, à droite, que les boîtes « standard » ne peuvent pas contenir de films ; c'est typique des modèles qui sont toujours infiniment lacumaires.

Exemple

heig-**vd**

Haute Ecole d'Ingénieurs et de Gestion
du Canton de Vaud

- Evaluation de la valeur économique des connaissances
 - Nécessité d'une approche quantitative



Journées de Méthodologies de Recherche, JDZ-JDD, 2.2.2011, HEIG-VD

70

Fig. 9.2. Une estimation quantitative des connaissances est nécessaire en pratique pour l'estimation à grande échelle de leur valeur économique

Exemple

heig-**vd**

Haute Ecole d'Ingénieurs et de Gestion
du Canton de Vaud

- Robotique et IA: Foot et aide domestique



Journées de Méthodologies de Recherche, JDZ-JDD, 2.2.2011, HEIG-VD

69

Fig. 9.3. A Singapour, un humanoïde, NAO, servait de médiateur entre humain et autres robots. Par ailleurs, notre robot RH-5 participait au test d'aide aux achats dans un centre commercial.

10. COMMANDE MULTIMODALE

En automatisation avancée, il est opportun d'aborder un mode original de commande pour les cas difficiles. Il s'agit, lorsqu'une commande, un régulateur "classiques", uniques, ne suffisent pas à atteindre l'objectif requis pour une application, d'en définir *plusieurs*, adaptés à des *contextes différents et complémentaires*. Puis de les exploiter tous simultanément, et d'en combiner les commandes, mais avec, pour chacun, une pondération liée à la proximité instantanée des conditions spécifiques dans lesquelles il a été défini.

L'approche s'applique ici à titre d'exemple, et de même au laboratoire, à un cas classique, représentatif de bien des processus industriels et plus généralement à beaucoup de systèmes naturellement instables : la commande d'un pendule inversé.

On se contentera de travailler en mode simulé, mais les mouvements et la dynamique du système seront rendus avec beaucoup de réalisme. En fait la partie commande reste tout-à-fait la même pour la commande des systèmes réels.

Principe

De façon générale, la documentation et le programme (codes source et exécutable) fournis dans le cadre du laboratoire permettent d'étudier, de pratiquer, et donc d'acquérir certaines compétences dans le domaine abordé.

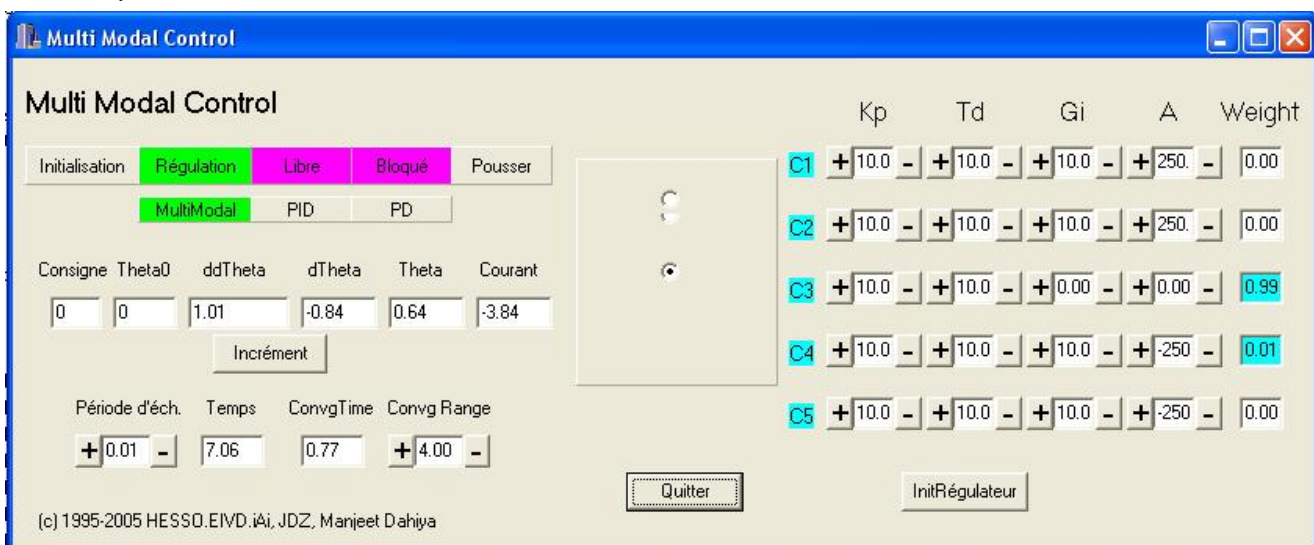
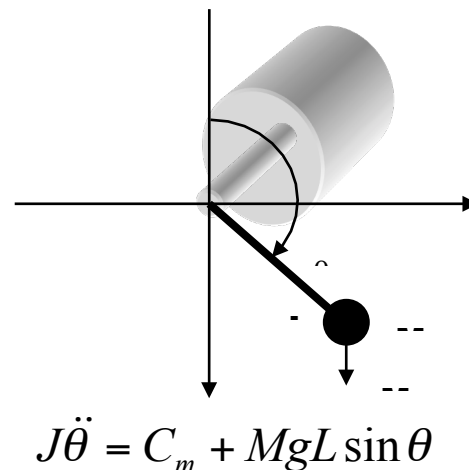


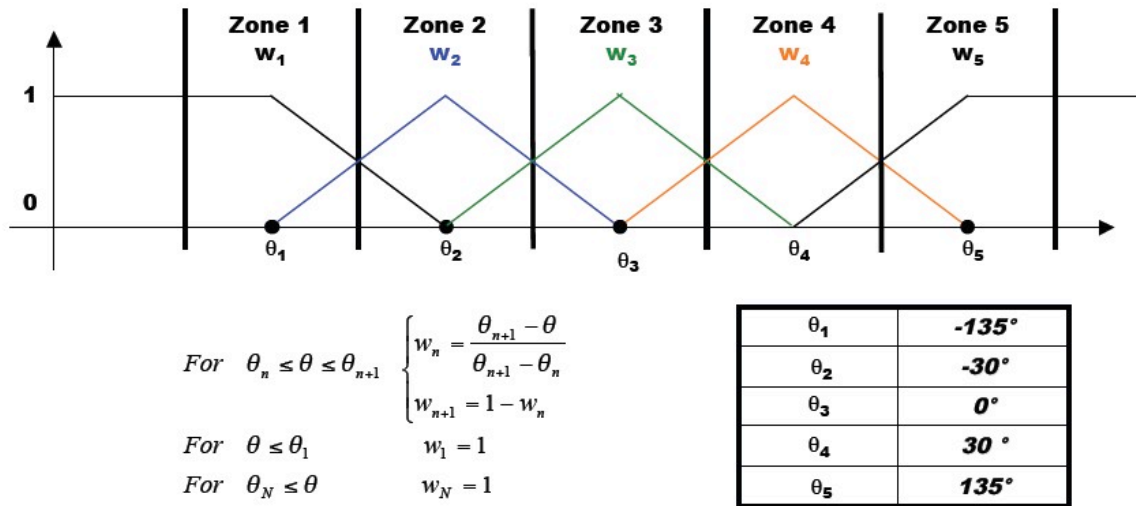
Fig. 10.1 Ecran de contrôle de la commande multimodale d'un pendule inversé disponible au laboratoire LaRA.

On y voit notamment qu'au lieu d'un seul régulateur classique pour l'ensemble du domaine de travail, par exemple, un PID, on aura ici 5 régulateurs, comprenant pour 4 d'entre eux (cas où la position de consigne n'est pas à la verticale de l'axe) une composante constante non nulle destinée à équilibrer localement la force (statique) due à la gravité.

L'idée majeure est que la commande multimodale, U_m , se calcule comme la moyenne de toutes les commandes élémentaires optimisées localement, U_n , chacune pondérée par sa fonction propre, w_n , en fonction de l'état courant où l'on se situe dans le domaine, $v(t)$.

$$U_m(t) = \frac{\sum_{n=1}^N w_n(t) \cdot U_n(t)}{\sum_{n=1}^N w_n(t)} = \frac{\sum_{n=1}^N w_n(v(t)) \cdot U_n(t)}{\sum_{n=1}^N w_n(v(t))}$$

Dans le cas présent, la variable physique est unidimensionnelle, il s'agit de l'angle du balancier, et la figure suivante y présente les fonctions de pondération adaptées à chaque domaine.



HESSO.HEIG-VD, J.-D. Dessimoz, 26.04.2010

19

Fig. 10.2 Fonctions de pondération pour 5 régulateurs dans un domaine unidimensionnel. La similitude avec les fonctions d'appartenance vues en logique floue est évidente, et vise de façon similaire également à passer de façon progressive d'un régulateur à un autre.

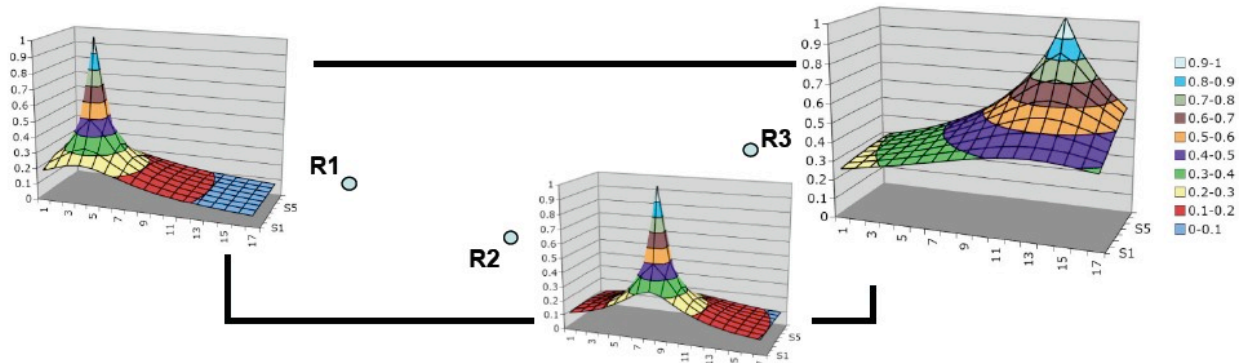
De façon similaire, la figure suivante illustre un cas où trois commandes élémentaires ont été dimensionnées dans un domaine de travail bidimensionnel, et les fonctions de pondération associées sont bidimensionnelles également.

- Amélioration de performances par régulateurs multiples
- Extrapolation de chaque régulateur R_i avec une surface de pondération en « cloche » (exemples: gaussiennes, ou ci-dessous fonctions de distances inverses W_i pour 3 régulateurs et un domaine 2-D, càd. à deux paramètres)
- Avec normalisation en tous points avec une division par la somme des poids des régulateurs en ces points

$$w_i = \frac{1}{k_i \cdot (1 + d_i)}$$

$$d_i(x, y) = \sqrt{(x - R_{ix})^2 + (y - R_{iy})^2}$$

$$w_i' = \frac{w_i}{w_1 + w_2 + w_3}$$



HESSO.HEIG-VD, J.-D. Dessimoz, 26.04.2010

27

Fig. 10.3 Fonctions de pondération pour 3 commandes élémentaires dans un domaine bidimensionnel..

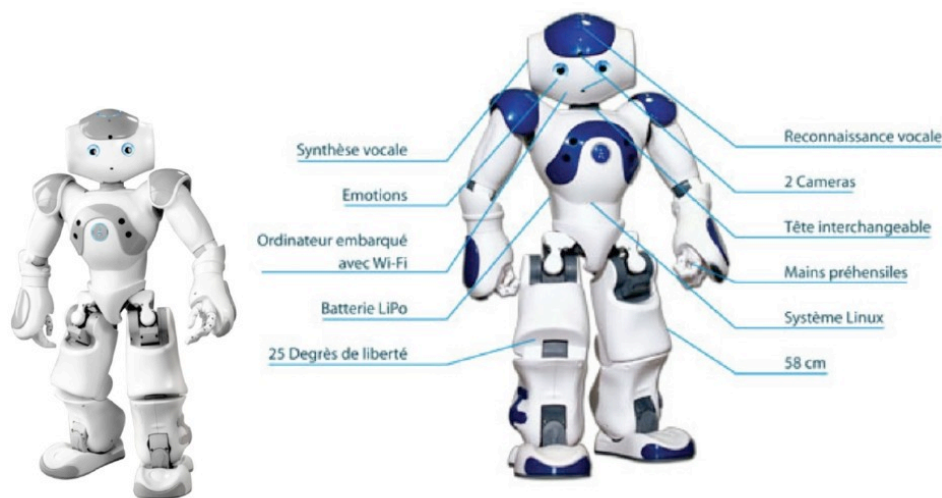
Notons que lorsque plusieurs fonctions de pondération (w) sont définies au même point d'un domaine de commande, il convient de les normaliser, ce qui se fait facilement, comme indiqué sur la figure précédente (w_i').

11. ROBOT HUMANOÏDE NAO

L'intelligence est parfois perçue comme indissociable du système qui l'implémente. Ceci est particulièrement évident pour le cas des systèmes dits « en situation ». Dans ce sens, un robot humanoïde nous offre une possibilité d'implémentation d'une intelligence artificielle conduisant à des résultats potentiellement similaires à celle de l'homme aussi. Le laboratoire LaRA dispose d'un tel robot, NAO, intégré chez nous à un groupe de robots comprenant nos propres prototypes, et ceci notamment avec coordination du système dans notre environnement Piaget.

NAO est un robot humanoïde français, de l'entreprise Aldebaran Robotics. En version académique, NAO a de riches ressources (capteurs, actionneurs, systèmes de calcul) et fonctionnalités (reconnaissance vocale et visuelle, locomotion et équilibre)

Le rôle de NAO, au laboratoire de robotique et automatisation (LaRa) de la HEIG-VD, est notamment d'assurer la médiation entre humains et robots coopératifs déjà existant au laboratoire ainsi qu'aux autres machines



- Cf. Sugamele et Taboada 2011, HEIG-VD.TIN.iAi.LaRA

HESSO.HEIG-VD, J.-D. Dessimoz, 5.04.2011

8

Les caractéristiques techniques de NAO sont les suivantes :

- Hauteur de 58cm pour 4.3kg
- Autonomie d'environ 90mn
- Possède jusqu'à 25 degrés de liberté pour la version académique
- Langage de programmation : C++, Python, Urbi
- Connectivité : Ethernet ou Wifi

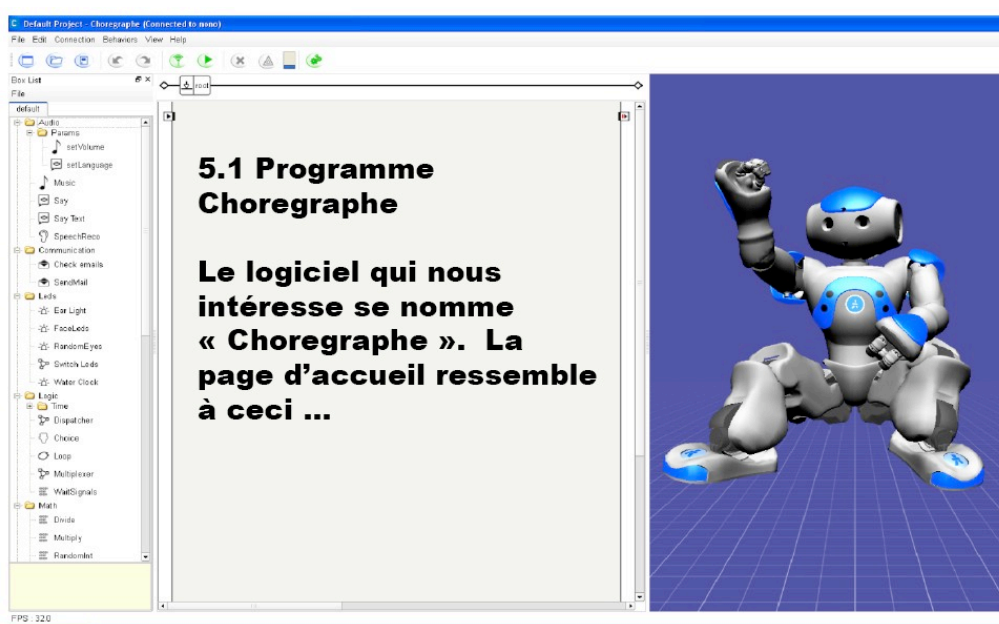
NAO: en chinois « cerveau », en japonais: « force », nom pour garçons et filles.

- Aldebaran en anglais: l'étoile la plus brillante
- Capteurs tactiles sur la tête
- Liaison infrarouge entre Naos (2 capteurs)
- 32 capteurs effet Hall
- 2 capteurs de force dans les semelles 0-25N
- 2 sonars (15-70cm)
- CAP Robotique , clusters
- 1 gyro 2 axes
- 1 accelerometer 3 axis

- 25 ddl total (moteurs Maxon)
- Geode 500 mhz (arm-9) depuis fin 2009
- Environ 500 plateformes vendues en 2010

Au niveau des capteurs, Nao est équipé d'une centrale inertielle 5 axes, de capteurs à ultrason et de capteurs de pressions sous les pieds. Il peut également communiquer avec un humain. Pour cela, il possède 4 microphones ainsi que 2 haut-parleurs. Pour la vision, il est équipé de deux caméras (CMOS, 30fps, 640*480 pixels).

Les aspects principaux de l'utilisation de NAO s'illustrent ci-dessous par une sélection des écrans de la présentation powerpoint du cours.



File Edit Connection Behaviors View Help



- 1 Permet de créer un nouveau projet
- 2 Permet d'ouvrir un projet existant
- 3 Permet de sauvegarder un projet en cours
- 4 Permet de revenir à l'étape précédente
- 5 Permet de passer à l'étape suivante
- 6 Permet de se connecter à Nao
- 7 Permet de lancer le programme attention, Nao bouge !
- 8 Permet d'arrêter le programme attention à la position dans laquelle se trouve Nao
- 9 Outil de débogage
- 10 Si en mouvement, indique qu'une tâche est en cours
- 11 Permet d'enclencher/déclencher les moteurs encore une fois, faites attention à la position dans laquelle se trouve Nao avant de déclencher les moteurs. Une position recommandable est celle où le robot est assis.

5.1.2 Mouvements en direct (interactif) ...suite

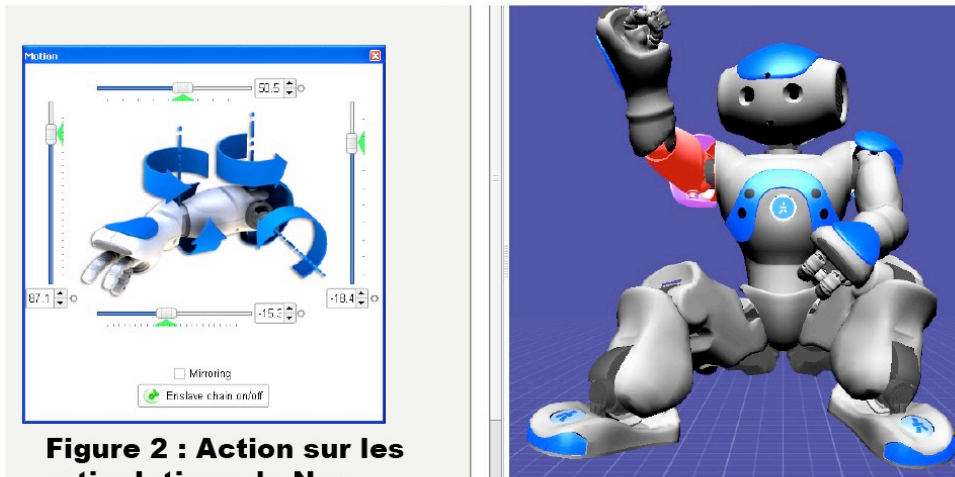
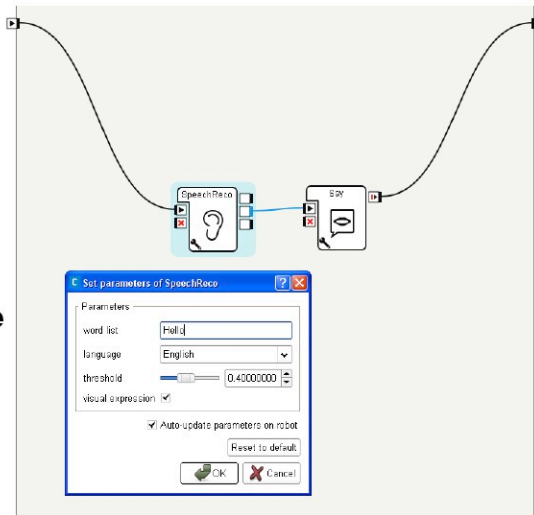


Figure 2 : Action sur les articulations de Nao

HESSO.HEIG-VD, J.-D. Dessimoz, 14 nov. 2009

19

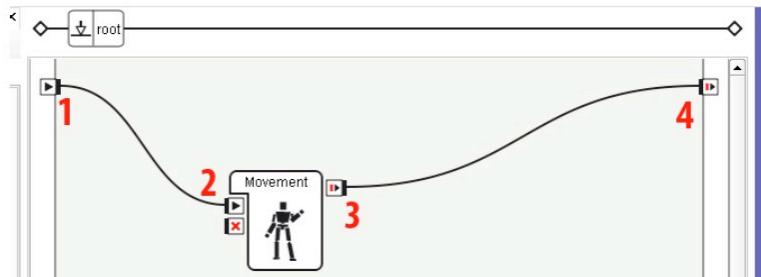
Figure 3 : Exemple de programmation sans mouvements



HESSO.HEIG-VD, J.-D. Dessimoz, 14 nov. 2009

22

- La programmation se fait de manière graphique en reliant le début du programme (1) à l'entrée du bloc (2) puis la sortie de ce même bloc (3) à la fin du programme (4).



HESSO.HEIG-VD, J.-D. Dessimoz, 5.4.2011

23

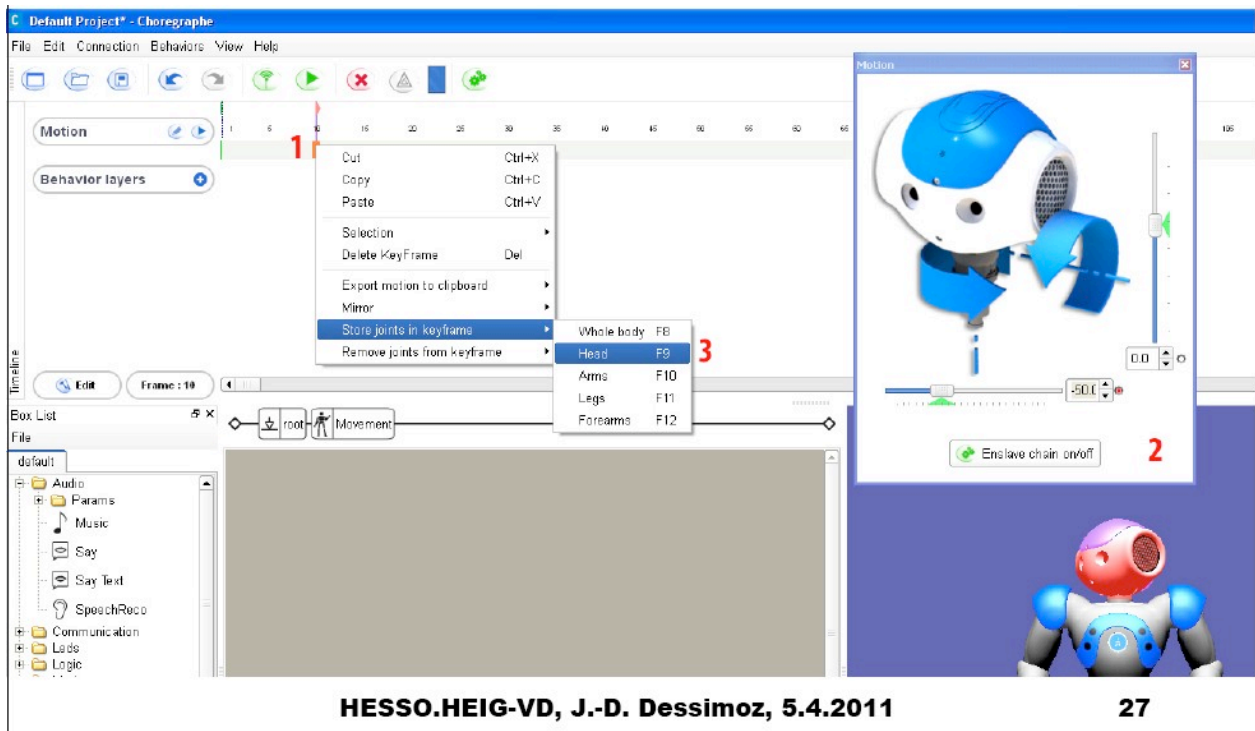
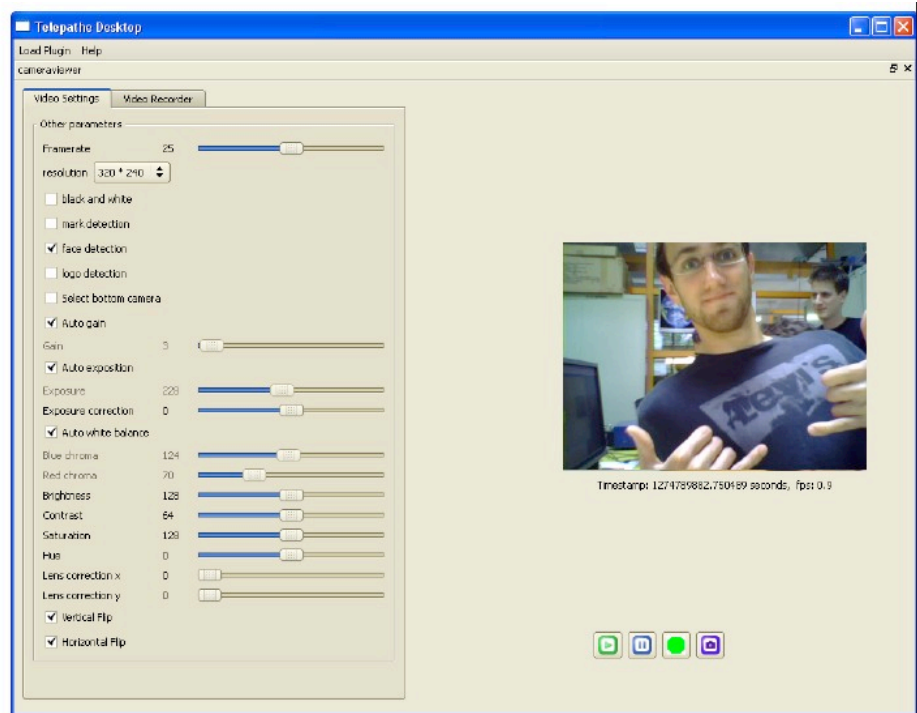










Figure 5
Prise de vue
au moyen
d'une des
caméras de
Nao



Les trois illustrations suivantes concernent la communication entr ordinateur et NAO, et plus particulièrement son intégration dans notre environnement Piaget.

-  **Connect** **Permet la connexion avec un ordinateur**
-  **Receive** **Attente d'une instruction provenant du PC connecté**
-  **Send** **Permet d'envoyer une information au PC connecté (utiliser un block «Text Edit » avant pour inscrire le message à envoyer)**
-  **Dispatcher** **Dans ce block il faudra mettre toutes les variables attendus de l'extérieur on pourra de cette manière, séparer chaque action voulue pour chaque variable inscrite:**

-  **SpeechReco mod** **Permet de réaliser une liste de mots que le robot sera capable d'entendre (Attention à configurer la langue parlé)**
Exemple:

-  **Gather** **Permet de rassembler les lignes séparées avec le « Dispatcher » en une ligne de sortie**

7.2 Recevoir et envoyer une information depuis l'interface du RH-Y

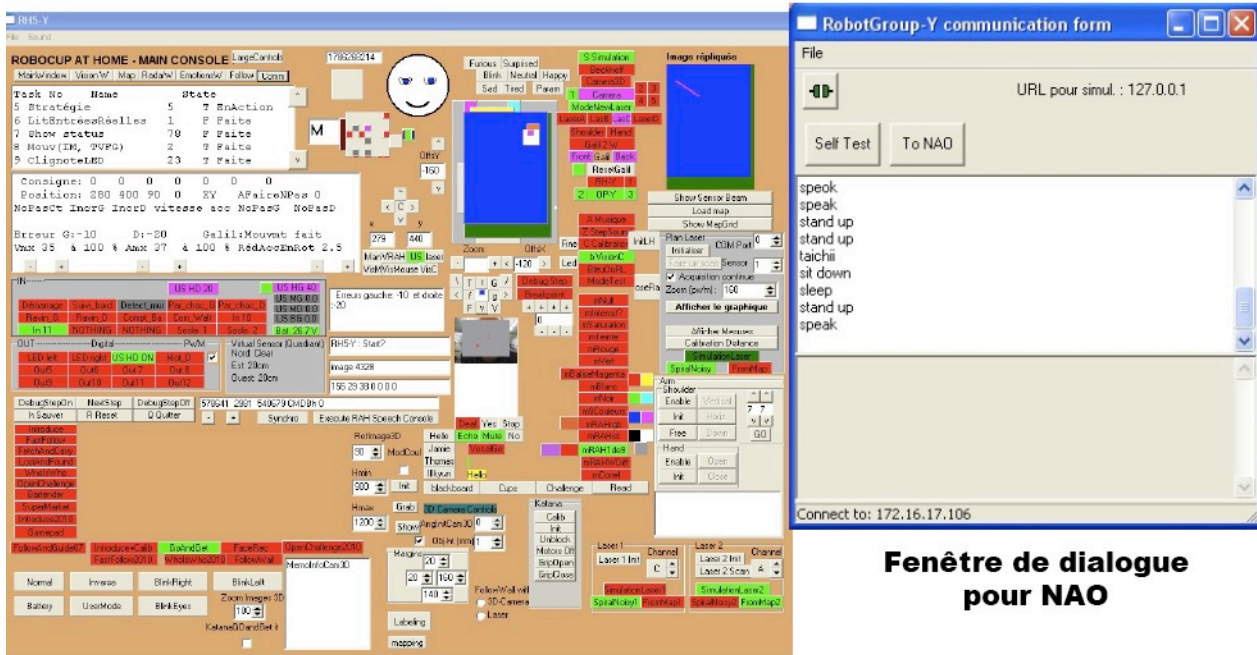
Il n'est pas nécessaire de réaliser un code dans le langage Piaget pour connecter l'ordinateur avec le NAO car le block « Connect » utilisé dans le code Choregraphe établi la connexion entre les deux. Toutefois, avant de commencer le code désiré en Piaget, il faut insérer la commande suivante qui ouvre la fenêtre de communication (les lignes inscrites sont des exemples) :

```
25699: ChatForm->Show();
      GoNext();
      break;case
```

Ensuite, il faut vérifier que la communication entre les deux robots soit établie. Pour ce faire on utilisera le code suivant :

```
25700: Answer = ReadMessageCom();
      if(Answer == "Ready");
      GoNext();
      break;case
      25703: NAOStringToSend = "INFO";
      GoNext();
      break;case
      ...
```

Une fois les deux instructions ci-dessus faites, on peut sans autre réaliser le programme voulu, toutefois, pour recevoir et envoyer une information il faudra utiliser les instructions suivantes :



Panneau de commande Piaget pour RH5-Y

Fenêtre de dialogue pour NAO

En conclusion et en bref, le robot NAO:

- est un humanoïde
- petit et léger (<5kg),
- avec de nombreux capteurs et moteurs
- et de nombreuses fonctionnalités
- intéressant pour communiquer en médiateur entre humains et machines
- Intégré à notre environnement multirobots Piaget

La présentation poerpoint du laboratoire sur NAO contient des informations supplémentaires ; et plus encore est disponibles dans nos rapports d'étudiants, la littérature scientifique, et dans la documentation du fabricant.

12. CONCLUSION

Le chapitre a présenté les bases de l'I.A. de deux manières complémentaires. D'une part, le domaine est présenté dans une perspective en quelque sorte historique, où se reflètent les diverses approches développées à ce jour et qui sont en somme classiques. Par ailleurs, des fondements théoriques ont été introduits, qui bien que moins connus, me paraissent pourtant beaucoup plus importants pour le futur. Le résumé extrême le plus pertinent est probablement le suivant:

- Utiliser une métrique pour les grandeurs cognitives
- Travailler dans de petits contextes
- Plus un modèle est bon (simple et conduisant à l'objectif), plus il est faux (incomplet par rapport à la réalité)

Une attention particulière a été apportée aux systèmes-experts. Cette technique, plus que d'autres relevant de l'I.A., a permis de résoudre certains problèmes pratiques, et apparaît encore valable pour résoudre certaines classes de problèmes.

Pour les robots, les techniques de l'I.A. apparaissent utiles pour générer de façon automatique des plans d'action ou pour définir de bonnes trajectoires. De façon plus singulière encore, elles devraient s'avérer précieuses en assemblage, pour l'élaboration des gammes d'assemblage et surtout le choix de gammes optimales.

Le problème majeur de l'intelligence artificielle est que la compréhension du monde et la communication avec l'homme ne peuvent se développer sans expérience vécue (y c. perceptions, émotions, histoire).

Le chapitre présente plusieurs systèmes à bases de connaissances développés au laboratoire de robotique de l'HEIG-VD dans le contexte de la manutention et l'assemblage.

Par son aspect extrêmement ouvert, le chapitre sur l'I.A. trouve bien sa place en fin du cours de robotique.

REFERENCES

[Bes] Didier Besset, "Ne me parlez plus de systèmes-experts. Faites-en un", Industrie et Technique, No 4, Févr. 1987.

[Des a] "Cognitics - Definitions and metrics for cognitive sciences and thinking machines", Jean-Daniel Dessimoz, Robotics Editions, Cheseaux-Noréaz, Switzerland, ISBN 978-2-9700629-1-2, pp169, January 2011, <http://cognitics.populus.ch>

[Des b] J.-D. Dessimoz, "Towards a theory of knowledge". Proc.3rd Ann. Conf. of the Swiss Group for Artificial Intelligence and Cognitive Sciences (SGAICO'91), Inst. for Informatics and Applied Mathematics, Univ. of Bern, Switzerland, Vol. IAM-91-004, pp. 60-71, 1992.

[Emm] S. Emmenegger, "Méthodes non conventionnelles de régulation; cas des algorithmes génétiques et pendule inversé", dipl. 2001, HESSO-HEIG-VD-iAi-LaRA, pp. 80 env.

[Far] H. Farreni, "Les Systèmes-Experts", Techniques avancées de l'automatisme, Cépadués Editions, pp.250, 1985.

[Gar] F. Gardiol, "Génération de plans d'actions pour la manipulation de cubes(Lisp)", rapp. de dipl., HEIG-VD, 1988.

[Gou] M. Gondran, "Introduction aux systèmes-Experts", Eyrolles, pp.100, 1986.

[HEC] "L'intelligence artificielle", Bull. HEC, dossier spécial, No 35, Univ. de Lausanne, Févr. 1991, pp.42

[Hof] R. G. Hoffman, "John McCarthy: Approaches to Artificial Intelligence", IEEE Expert, pp. 87-89, June 1990.

[Hun] V.D.Hunt, "Artificial intelligence and expert systems sourcebook", Advanced industrial technology Series, Chapman & Hall, London, New York, pp.315, 1986.

[Kap] G. Kaplan, "Industrial electronics", IEEE Spectrum, New York, pp. 47-48, Jan. 1992.

[Mas] P. Masson, "Définition automatique de points de prise pour un robot industriel doté de vision", dipl. 2001, HESSO-HEIG-VD-iAi-LaRA, pp. 60 env.

[Mic] D. Michie, "On Machine Intelligence", 2nd Ed., Ellis Horwood Ltd, Chichester, W Sussex, England, pp. 270, 1986.

[Nil] Nils J. Nilsson, "Principles of Artificial Intelligence", Symbolic Computation Series, Springer Verlag, Berlin, New York, pp.476, 1982.

[Pas] G. Pask, "Conversation, cognition and learning", Elsevier, Amsterdam, 1975.

[Per] Claude Pernollet, "Système-expert pour l'aide à l'installation de robots", rapp. de dipl., HEIG-VD, 1988.

[Sch] Rolf Schweitzer, "Système-expert pour la manipulation de cubes (Prolog)", rapp. de laboratoire., HEIG-VD, 1985.

[Sol] M. Solbes, "Système-expert pour la manipulation de cubes", rapp. de laboratoire, HEIG-VD, 1984.

ANNEXE - ARCHIVES - SYSTEMES-EXPERTS

Les systèmes-experts constitue l'une des contributions de l'I.A. au génie informatique. Ils se caractérisent par un certain impact pratique et un certain succès commercial. Ce paragraphe est pour l'essentiel tiré de [Bes].

Pour Ed. Feigenbaum, les systèmes experts constituent le moyen par lequel les chercheurs en intelligence artificielle publient leurs résultats sous une forme assimilable par l'industrie. Feigenbaum oppose donc l'intelligence artificielle qui est une activité de recherche pure et les systèmes experts qui sont, avant tout, des réalisations pratiques. Par conséquent, loin de représenter l'état de l'art de la recherche, ils utilisent des méthodes qui ont fait leurs preuves au sein de la communauté académique depuis quelques années. En effet, les premiers systèmes experts ont été développés dans la première moitié des années 1970.

Les systèmes experts sont des programmes d'ordinateurs qui résolvent des problèmes en raisonnant sur des symboles au lieu d'effectuer des calculs. Ils ont été qualifiés d'experts car les premiers de ces programmes s'attaquaient à des problèmes que seuls des experts étaient capables de résoudre. Puisque les systèmes experts sont utilisés pour résoudre des problèmes à caractère non numérique, le résultat d'un tel programme sera l'évaluation d'une situation ou la solution d'un problème. Dans ce cas, au lieu de parler d'exécution d'un programme, on dit que l'on consulte le système. Cela vient peut-être de ce que les premiers systèmes experts exerçaient leurs talents dans le domaine médical. Le nom de consultation a été généralisé à tout système expert.

Dans un système expert, le problème est caractérisé par un ensemble de propriétés. Si la description du problème peut se faire sous forme de paramètres numériques, on se retrouve généralement dans le domaine de la recherche opérationnelle. Dans le cas d'un système expert, ces propriétés sont données principalement sous une forme qualitative. Bien entendu, certaines de ces propriétés peuvent être à caractère numérique. L'ensemble des propriétés constitue ce qu'on appelle une base de connaissance. La solution d'un problème sera obtenue en utilisant des méthodes de raisonnement basées sur la logique. La partie du programme chargée d'effectuer le raisonnement est dénommée le moteur d'inférence. Cette capacité que possède un système expert de raisonner sur de la connaissance constitue la différence fondamentale par rapport à un programme classique. La figure 1 montre une représentation graphique de l'architecture d'un système expert.

Les systèmes experts: avant tout une solution informatique

En fait, on peut très bien fabriquer un système «simplet», fonctionnant sur les mêmes principes qu'un système dit «expert», afin, par exemple, de décharger les employés de problèmes simples mais fastidieux. Aussi, au lieu de parler de systèmes experts, je préfère utiliser le terme de systèmes à base de connaissance (knowledge based systems) par opposition aux systèmes à base algorithmique que sont les programmes d'informatique classique. Les termes

d'intelligence artificielle et de systèmes experts ont une connotation trop anthropocentrique. Cela a déjà nuit aux ordinateurs à l'époque où on les appelait cerveaux électroniques. De cette appellation, les gens ont développé une peur superstitieuse de la machine et cette attitude est en train de se développer à l'égard des systèmes experts.

Il ne faut pas perdre de vue qu'un système à base de connaissance est avant tout un programme d'ordinateur. Certes, la méthode utilisée par un tel programme est nouvelle. Grâce à cette approche, de nouveaux problèmes, qui jusque-là ne pouvaient être résolus sur ordinateurs, deviennent accessibles. Cela permet même d'aborder d'anciens problèmes avec un point de vue différent. Mais il ne faut pas perdre de vue que le but principal d'un ordinateur est d'aider les gens à trouver des solutions à des problèmes complexes ou fastidieux.

Par exemple H. Simon et son groupe à l'université de Carnegie-Mellon ont développé un environnement intéressant pour l'écriture de systèmes-experts (bases de règles). Pour la configuration de leurs ordinateurs, Digital Equipment avaient notamment écrit des systèmes-experts contenant plus de 6000 règles.

Dans le futur il faut s'attendre à trouver des systèmes à base de connaissance fonctionnant de concert avec des programmes à base algorithmique. Les utilisateurs ne seront plus conscients de la différence, de même qu'aujourd'hui, les personnes qui se servent d'un tableur et d'un traitement de texte ne vérifient plus si les programmes qu'ils utilisent sont écrits selon la méthode de programmation structurée ou non. L'important pour l'utilisateur réside dans la capacité d'un programme à résoudre un problème donné, et dans sa performance, autant sur le plan de la rapidité que celui du confort d'utilisation

Dans les deux prochaines sections, les deux composantes principales d'un système à base de connaissance, les représentations de la connaissance et les méthodes d'inférence, seront décrites succinctement afin de pouvoir mettre en évidence les différences entre ces systèmes et ceux de l'informatique classique.

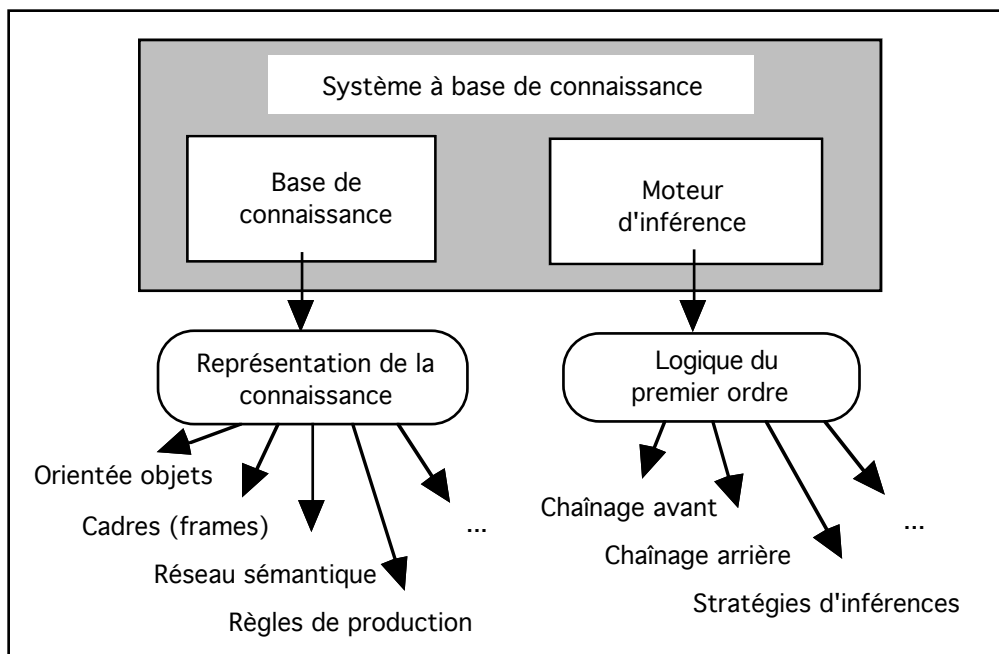


Fig 1.—Composants d'un système à base de connaissances

A.1. Les systèmes-experts

Depuis quelques années, les chercheurs ont perçu qu'une composante essentielle manquait à tous les programmes existants: la connaissance approfondie du domaine concerné, le savoir de l'expert reposant sur la connaissance accumulée au fil des années, sur l'expérience.

Et si on veut aller plus loin, faire des analogies, des recoupements, exprimer plus largement son intelligence, il faut une masse importante de *connaissance* dans des domaines très divers.

Encore faut-il *savoir s'en servir* et on voit déjà le mal que l'on a avec les bases de données lorsque l'on veut simplement extraire des faits de la base. Dans les *systèmes experts*, on se propose de rechercher des faits qui n'ont pas été- enregistrés dans la base des faits et que l'on pourra déduire des faits de la base.

Le but d'un système expert est de modéliser le comportement d'"un expert humain accomplissant une tâche intellectuelle dans un domaine précis". Les systèmes experts se placent à la jonction des deux approches de l'I.A., la *représentation des connaissances* d'une part, la *démonstration automatique* d'autre part.

Ils seront donc formés de deux parties indépendantes: une base de connaissance, un démonstrateur de théorèmes (ou moteur d'inférence).

La base de connaissance traduit dans un *langage déclaratif et modulaire* des connaissances expertes d'un domaine. Le démonstrateur est chargé *d'appeler et d'utiliser celles-ci à bon escient*, pour répondre à une question ou pour résoudre un problème.

Deux caractéristiques essentielles différencieront un système expert d'un programme informatique ordinaire; à tout moment, le système peut:

- expliquer son comportement à l'expert par la trace des connaissances utilisées dans le raisonnement,
- recevoir de l'expert de *nouvelles connaissances sans aucune nouvelle programmation* par simple mise à jour de la base de connaissances.

Pour cela, la base de connaissance doit être lisible, indépendante du moteur d'inférence (mais interprétable par lui), *conçue par et pour l'expert*.

A.2 Représentations de la connaissance

Un programme d'intelligence artificielle utilise le calcul symbolique, La représentation de la connaissance se fait sous la forme de structures de données et de leurs procédures d'utilisation. C'est la combinaison des données et des procédures qui donne à un programme un comportement qui, sur un domaine restreint, peut apparaître comme intelligent. Par analogie au systèmes de base de données, on appelle une *base de connaissance* la partie dans laquelle on mémorise la connaissance.

Il existe de nombreux formalismes de représentation de la connaissance. Mais, en principe, on est capable de représenter toute forme de connaissance dans chaque formalisme. Bien entendu, certaines représentations sont mieux adaptées que d'autres à un problème donné. La figure 2 donne un aperçu de l'équivalence entre plusieurs modes de représentation décrits ci-après.

Quel que soit le mode de représentation, un système à base de connaissance, comme tout programme d'intelligence artificielle, utilise le calcul symbolique. En informatique classique, on parle aussi de symboles, mais ceux-ci correspondent seulement à des emplacements mémoire. Un programme classique est aussi capable de manipuler des symboles sous leur forme codée. En effet, un nom est représenté comme une chaîne de chiffres qui correspondent aux codes de chaque caractère le composant. Cependant, la signification du nom reste hors de portée du programme.

En intelligence artificielle, on a très vite éprouvé le besoin de manipuler des symboles en plus de nombres. Pour pouvoir le faire, il faut attacher au nom du symbole un mode d'utilisation que le programme saura exécuter lorsqu'il doit travailler avec ce symbole. On ne peut pas vraiment dire que le programme connaît la signification d'un symbole mais il connaît le contexte dans lequel ce symbole peut être utilisé.

Par exemple, dans un programme chargé de piloter un robot mobile, on définira une fenêtre et une porte comme des objets que l'on peut ouvrir ou fermer: en plus on indiquera que la porte

peut être franchie lorsqu'elle est ouverte. Pour le robot, cette propriété lui permettra de différencier une porte d'une fenêtre. Les autres nuances lui resteront inconnues.

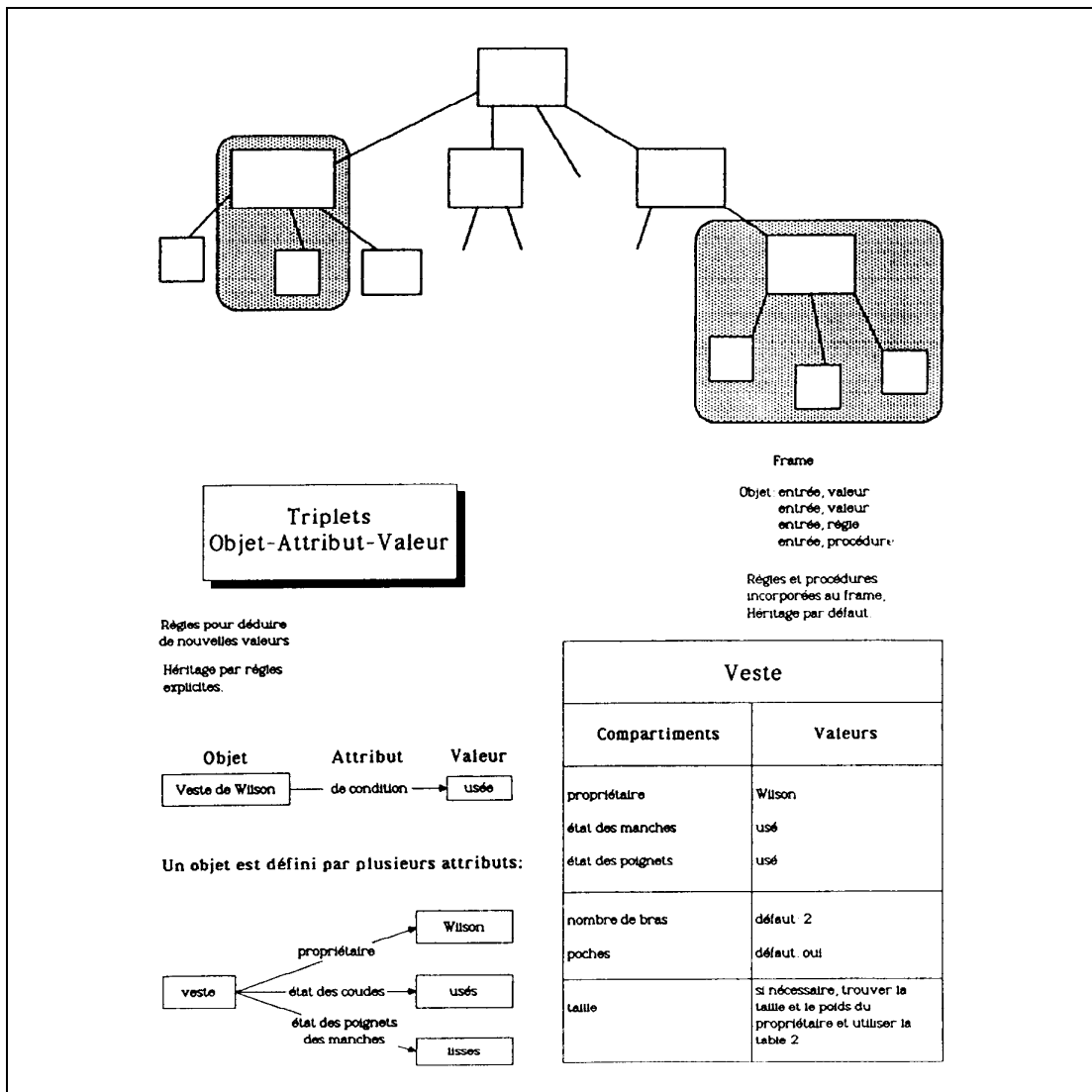


Fig 2.—Dualité des formalismes de représentation.

A.2.1 Représentation par objet

Une possibilité pour représenter le mode d'utilisation d'un symbole est d'établir une liste des propriétés de l'objet représenté par ce symbole. La manipulation sur les objets sera effectuée en fonction de leurs propriétés. On parle de représentation par objets (object oriented representation).

Considérons par exemple un système chargé de faire des recommandations sur les travaux nécessaires à la fabrication d'une pièce en mécanique. Ce système aura connaissance de trois objets: les pièces à fabriquer, les machines qui peuvent être utilisées et enfin les diverses opérations que l'on peut réaliser. Ce dernier objet montre qu'il n'y a pas nécessairement correspondance avec un objet physique. Une pièce sera caractérisée par sa forme, ses dimensions et sa composition, une machine par sa tête de travail et ses ajustements, une opération par la ou les transformations géométriques qu'elle fait subir à une pièce.

Le concept de propriété d'un objet peut se généraliser. Cela peut être un nouvel objet attaché au premier, une procédure à effectuer lorsque cet objet est sous investigation ou finalement une stratégie de recherche pour certaines propriétés de l'objet considéré.

En poursuivant l'exemple précédent, on va définir des relations qui associent à chaque opération un certain type de machine et certains paramètres de la pièce à une série d'opérations à effectuer dans un certain ordre. Certains paramètres de machine peuvent être obtenus par défaut: si on ne dit rien, le lubrifiant sera de l'huile ordinaire. D'autres seront obtenus en fonction de propriétés déjà établies: la vitesse de tournage se détermine en fonction d'une table dont les entrées sont la dureté du matériau et l'outil de coupe. Lorsque l'on rattache un objet à un autre (on parle alors de structure hiérarchique), on peut définir que certaines propriétés sont transmises de l'objet parent à son descendant. On dit que l'objet hérite des propriétés de son parent. L'héritage automatique de propriétés est une des caractéristiques des schémas (frames).

Ainsi, au lieu de définir qu'une forme conique doit être fabriquée sur un tour, de même qu'une forme cylindrique, on va introduire le concept de formes à symétrie de révolution qui constituera un objet. Comme propriété de cet objet, on indiquera que ce type de pièce doit être fabriqué sur un tour. Puis, on définira que les pièces à forme conique ou cylindrique sont des formes à symétrie de révolution.

A.2.2 Réseaux sémantiques

Plutôt que de mettre l'accent sur les objets, il est possible de mettre en évidence les relations qui existent entre ces objets. Les relations peuvent être à caractère statique (appartenance, hiérarchique) ou dynamique (relations de cause à effet, actives ou passives). On parle alors de réseaux sémantiques (semantic networks). Les réseaux sémantiques sont surtout utilisés dans les problèmes de compréhension du langage.

Représentation de l'incertitude

Dans les problèmes de décision, il existe de nombreux cas où une certaine propriété ne peut pas être établie avec certitude. Par exemple, la prédisposition d'un individu au diabète se mesure avec une probabilité qui dépend de ses antécédents héréditaires et de sa condition physique actuelle. Plusieurs systèmes offrent donc la possibilité de pondérer chacune des valeurs utilisées par un degré de certitude qui varie entre 1 (certitude absolue) et -1 (rejet catégorique). Les valeurs intermédiaires sont laissées à l'appréciation du concepteur de la base de connaissance, l'échelle de valeurs étant moins significative que la façon dont se combinent deux conclusions ayant des valeurs semblables pondérées par des facteurs de certitude différents. La loi de combinaison la plus simple est celle utilisée dans Mycin, qui calcule le facteur résultant comme s'il s'agissait de probabilités d'événements indépendants l'un de l'autre. Puisque dans le cas général, l'indépendance de l'information ne peut être garantie, cela revient à faire une approximation.

Quelques systèmes utilisent des approches beaucoup plus exactes au prix d'une complexité accrue. Par exemple, dans la théorie de Dempster-Shafer on a besoin de deux coefficients pour mesurer l'incertitude. Une étude récente en logique probabiliste utilise des matrices à plusieurs dimensions.

Une autre façon d'aborder la représentation de l'incertitude consiste à utiliser la logique floue (fuzzy logic). Il s'agit ici de représenter la transition entre deux valeurs contradictoires (par exemple oui ou non, enfant ou adulte...) de manière continue. La logique floue est une théorie mathématique passionnante, mais son utilité dans les systèmes à base de connaissance reste à être démontrée car elle ne résout pas le problème d'une incertitude a posteriori. Il est souvent beaucoup plus efficace de représenter explicitement l'équivoque d'une situation.

La représentation de l'incertitude a été longtemps considérée comme l'un des fondements des systèmes à base de connaissance. Aujourd'hui, cependant, les avis sont partagés et certains chercheurs sont revenus de leur enthousiasme de la première heure. Si vous avez vraiment besoin de représenter l'incertitude, il vaut mieux utiliser une représentation simple en restant conscient de ses limitations que de se lancer dans une théorie complexe.

A.2.3 Règles de production

Les relations logiques entre les éléments de connaissance introduits dans un programme s'écrivent en général à l'aide de règles de production. Une règle de production possède un ensemble de propositions logiques qui forment les conditions nécessaires pour la déduction d'autres propositions. Dans le cas de réseaux sémantiques utilisant des relations causales, cela n'est pas nécessaire puisqu'une règle de production est équivalente à une relation de cause à effet.

Voici un exemple de règle:

Si on observe des courbes irrégulières sur l'oscilloscope
et que l'alternateur ne répond pas correctement
alors le régulateur de tension est défectueux.

Les conditions d'une règle sont aussi appelées les prémices. En général une règle peut avoir plusieurs conclusions. Les règles sont évaluées en utilisant les principes de base de la logique élémentaire. La partie du programme chargée de cette évaluation est appelée moteur d'inférence.

A.3 Méthodes d'inférence

Le moteur d'inférence est la partie du programme qui effectue les déductions pouvant être effectuées à partir des faits existants à un instant donné dans le programme. Dans le cas d'un réseau sémantique, l'inférence est intimement liée à l'établissement du réseau lui-même, c'est-à-dire des relations qui existent entre les différents objets connus du système. Pour les autres formalismes de représentation de la connaissance, on se sert le plus souvent de règles de production. Dans ce cas, on utilise généralement un raisonnement fondé sur le *modus ponens* qui se définit comme suit:

Si A implique B
et A est vrai
alors B est vrai.

Il ne s'agit là qu'une des deux possibilités de raisonnement offertes par la logique élémentaire. Peu de systèmes font appel au *modus tollens* ou raisonnement par l'absurde. Pour effectuer le raisonnement, il reste à définir une méthode pour la recherche des règles à utiliser. Il n'y a que deux méthodes possibles: le chaînage avant et le chaînage arrière.

A.3.1 Chaînage avant

Le chaînage avant consiste à rechercher dans la base de connaissance toutes les règles dont les prémices sont satisfaites par les faits en mémoire. De ces règles, on déduit de nouveaux faits. A partir de ces nouvelles informations, on peut, à nouveau, rechercher des règles qui vont se trouver satisfaites grâce au nouvel ensemble de faits. Cette procédure est répétée jusqu'au moment où il n'y a plus de nouvelles déductions possibles. Cette méthode porte le nom de chaînage avant parce que la direction du raisonnement utilise les données comme point de départ pour parvenir au résultat.

Le chaînage avant a l'avantage d'utiliser la totalité des faits présents en mémoire à un instant donné. Il va explorer toutes les implications de ces déductions. Dans certains cas, cela peut se révéler assez inefficace, puisque des voies sans issue seront étudiées sans possibilité de pouvoir les rejeter à priori. Par contre, puisque les informations peuvent être fournies au programme sans ordre précis, le chaînage avant se prête bien au traitement de données en temps réel.

Règle 1 Si moyen = conduire et lieu = ville alors action = taxi	Règle 2 Si moyen = conduire et lieu = pas en ville alors action = conduire
Règle 3 Si moyen = marcher et temps = mauvais alors action = imperméable et marcher	Règle 4 Si moyen = marcher et temps = bon alors action = marcher
Règle 5 Si distance > 5 km et lieu=ville alors action = taxi	Règle 6 Si distance > 1 km et délai < 15 min alors moyen = conduire
Règle 7 Si distance > 1 km et délai > 15 min alors moyen = marcher	Règle 8 Si distance < 1 km alors moyen = marcher

Fig 3. A Exemple — Règles de production.

A.3.2 Chaînage arrière

Pour raisonner par chaînage arrière, on se donne une proposition à prouver. Le programme cherche toutes les règles qui effectuent une conclusion à ce sujet. Les prémices de chacune des règles seront testées. Si une des propositions parmi les prémices de la règle ne se trouve pas dans les faits en mémoire, le programme va essayer de prouver cette proposition à son tour. Ce mécanisme va déclencher l'enchaînement des règles.

Question 1 Quelle est la distance?	Réponse 1 distance = 2 km
Question 2 Combien de temps avant la réunion?	Réponse 2 délai = 2 min.
Question 3 Où se trouve la réunion?	Réponse 3 lieu = ville

Fig 3 .B Exemple — Consultation.

Une consultation effectuée par chaînage arrière a l'avantage d'être orientée vers un but précis. En conséquence, l'ordre des questions posées à l'utilisateur du système semble plus naturel. Le chaînage arrière convient très bien aux problèmes de sélection, de diagnostic et de planning structuré. En général, cette méthode se prête mieux aux consultations interactives.

Règles satisfaites		Faits en mémoire
		distance = 2 km délai = 10 min. lieu = ville
1er passage	Règle 6 ->	moyen = conduire
2ième passage	Règle 1 ->	action = taxi
3ième passage		pas de nouvelle conclusion

Fig 3 . C Exemple — Chaînage avant

La figure 3 permet de comparer les deux méthodes sur un exemple où il s'agit de trouver la meilleure action à suivre pour se rendre à une réunion en fonction de la distance, du délai et du lieu. Dans un cas aussi simple, la base de connaissance n'utilise que le formalisme des règles de production (et bien entendu une représentation symbolique).

Dans le cas du chaînage avant, les règles satisfaites sont indiquées pour chaque itération avec les faits auxquels elles concluent. Pour le chaînage arrière, on a indiqué dans des cartouches les propositions à trouver (ponctuées par un point d'exclamation pour mettre en évidence la recherche), dans des ovales les règles utilisées par cette recherche et dans des rectangles les conclusions obtenues, soit par conclusion d'une règle, soit que le fait était connu du programme.

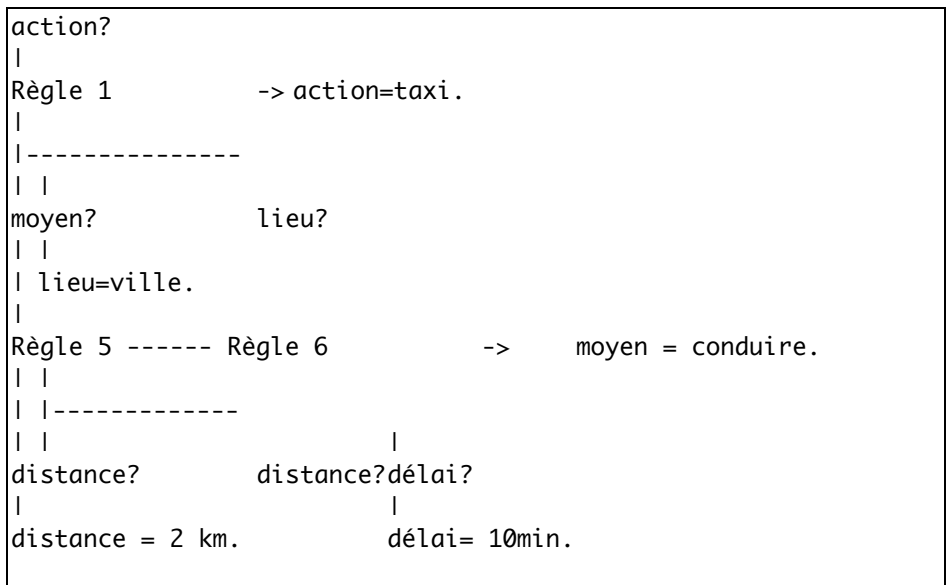


Fig 3. D Exemple — Chaînage arrière.

A.3.3 Stratégies d'inférence

Dans les deux méthodes décrites auparavant, on peut apporter quelques raffinements sur la façon la plus efficace de choisir les règles. Le choix d'une stratégie d'inférence s'appelle parfois résolution de conflit (conflict resolution). Il s'agit là d'un domaine très vaste. Par conséquent, nous nous limiterons à présenter deux exemples.

Dans le cas du chaînage arrière, il y a en général plusieurs règles qui peuvent avoir une conclusion similaire. De sorte que lorsque l'on cherche une valeur, on peut se poser la question de savoir quelles sont les règles à tester en priorité. La plupart des systèmes existants prennent simplement les règles dans l'ordre que le programmeur a choisi. Cependant, le choix peut

dépendre du contexte et certains systèmes offrent la possibilité de classer les règles selon plusieurs catégories.

Le problème est très similaire pour le chaînage avant. On peut laisser le système essayer toutes les règles dans la séquence choisie par le programmeur. Pourtant, dans certains cas, il peut se révéler plus efficace d'examiner en premier les règles les plus restrictives (c'est-à-dire celles qui comportent le plus de prémices) et de recommencer cette recherche chaque fois qu'une nouvelle conclusion a été établie.

A.4 Les outils de développement

Puisqu'un système à base de connaissance est un programme d'ordinateur, il est possible de l'écrire dans n'importe quel langage. Cependant, de même qu'il ne serait pas réaliste d'écrire un système de comptabilité ou de CAO en assembleur, il n'est pas efficace de construire un système à base de connaissance dans un langage de programmation classique. Des langages permettant le calcul symbolique ont été développés très tôt. Cependant, un langage tel que Lisp ne possède pas de moteur d'inférence ni de formalisme de représentation de la connaissance: Prolog ne possède qu'un moteur d'inférence. Pour pallier ces carences, on a construit sur ces langages des environnements qui possèdent un ou plusieurs moteurs d'inférence et des formalismes de représentation de la connaissance qui ont été construits sur ces langages (surtout sur Lisp). Enfin, il existe des outils de développement dans lesquels le moteur d'inférence et la représentation de la connaissance sont intégrés dans un nouveau langage et un environnement de développement (cross-référence, debugger, tracing, etc.). La *figure 4* montre la progression des langages et outils comparés aux couches de la programmation classique.

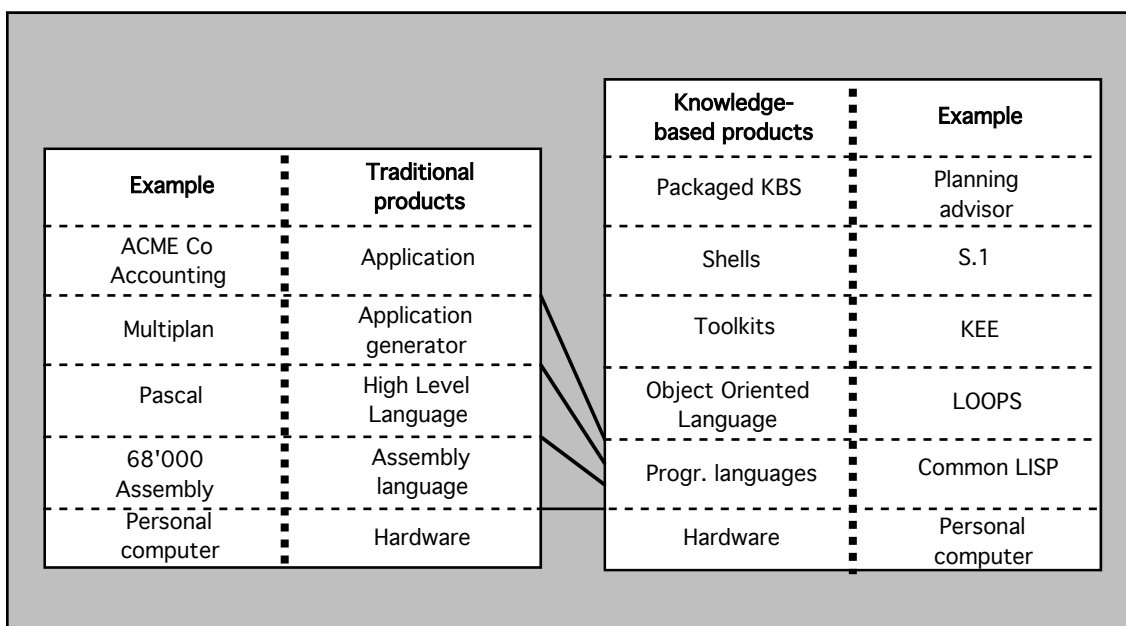


Fig 4.—Niveaux de réalisation.

Aujourd'hui, il existe des outils de développement pour systèmes experts sur presque tous les ordinateurs (voir encadré). La machine sur laquelle le développement doit s'effectuer peut être aussi modeste qu'un PC mais, bien entendu, la productivité de l'ingénieur de la connaissance s'en ressentira. Par contre, un système à base de connaissance modeste n'a pas besoin d'être réalisé sur un matériel aussi sophistiqué qu'une machine Lisp. Le développement d'un projet modeste sur PC permet à une entreprise de se familiariser aux techniques des systèmes à base de connaissance pour un budget limité. Bien plus important est la faculté de faire exécuter le système en production sur le hardware existant. Malheureusement, seuls quelques produits permettent cette possibilité. Grâce aux outils de développement actuels, le temps de réalisation d'un système à base de connaissance a été considérablement réduit. Vers 1975, il fallait 15

années-homme pour construire un système à base de connaissance en Lisp, une bonne partie du temps étant dédiée à créer un environnement de développement. Récemment, la compagnie Framatome a mis au point un système en utilisant l'outil M.1 en 9 mois de deux personnes (l'expert et l'ingénieur de la connaissance). Ce système, TIG, tourne sur un ordinateur personnel et est maintenant utilisé quotidiennement dans une usine.

La plupart des produits actuellement sur le marché sont des environnements bâtis autour d'un langage de base (Lisp, Prolog ou OPS 5). Puisqu'ils utilisent des langages de bas niveau, ils sont très flexibles, mais il faut du temps avant de pouvoir maîtriser l'outil. De plus, il est toujours nécessaire de connaître le langage de base. Le prix d'un produit varie énormément, de quelques centaines de francs à des centaines de mille.

En règle générale, les prix sont proportionnels aux possibilités offertes par un produit et donc à sa complexité. Des outils très bon marché tournant sur PC, très faciles à maîtriser, ne pourront résoudre qu'un nombre limité de problèmes. Par contre, les produits de haut de gamme (100'000 francs et plus) permettent de réaliser des représentations complexes et des inférences sophistiquées mais requièrent une bonne année de formation au minimum. Leur utilisation est justifiée par un projet de recherche.

Néanmoins, il ne faut pas perdre de vue que l'investissement dans l'acquisition d'un outil de développement ne représente qu'une fraction de l'investissement total de la réalisation d'un système à base de connaissance. Il faut prendre en considération, dans le coût de développement, le temps de l'ingénieur de la connaissance et celui de l'expert. Selon la taille du projet, l'investissement en programme et en matériel représente environ 10 à 20 % du coût final. Si l'on prend, comme exemple, un système qui permet le dépannage d'une machine-outil complexe livrée à l'étranger, on estime que le prix du développement est couvert en moins de deux ans par l'économie des déplacements du technicien de dépannage.

Malgré les nombreuses possibilités de représentation, l'enregistrement de la connaissance est un processus délicat et fastidieux. Pour programmer un système médical destiné à assister un généraliste, il faut emmagasiner l'équivalent de sept années d'études, plus tout le bon sens que tout être humain acquiert durant ses jeunes années et enfin toute l'expérience pratique qui distingue le spécialiste chevronné du jeune médecin fraîchement sorti de l'université. Il est clair que les méthodes actuelles de représentation de la connaissance sont encore trop primitives pour aborder un sujet aussi riche.

A.5 Progiciel de génération de systèmes-experts

Personal Consultant Plus, de Texas Instruments. Progiciel très efficace pour l'écriture de systèmes à bases de règles, avec traduction en langage naturel automatique des règles, contrôle exhaustif des valeurs de paramètres, représentation graphique des arborescences de règles, domaines multiples de connaissances, et aides diverses.

```

;si le cube de la pile de depart n'est pas libre [on ne peut pas le
;deplacer car il y a des cubes au-dessus], il faut le rendre libre
(if (>
  (- (length
      (list-ref probleme
        (send (eval cube-a-deplacer) get-pile)
      )
    )
    1 )
  (send (eval cube-a-deplacer) get-niveau)
)

;then
(set! possible
  (regles (list-ref
          (list-ref probleme
            (send (eval cube-a-deplacer) get-pile)
          );sélection de la pile
          (+ 1 (send (eval cube-a-deplacer) get-niveau))
          ;sélection du cube au-dessus
        )
    pile-consigne niveau-consigne
  ) ;regles
);set!
);if

(if possible
  ;then
  ; maintenant, on a un cube a deplacer
  ;il faut le poser quelque part
  (set! pile-destination
    (cherche-destination niveau-consigne
      pile-consigne
      (send (eval cube-a-deplacer) get-pile)
    )
  );set!
);if

```

Fig.21 Extrait du module Lisp "Règles.s": cette règle correspond à l'exemple du système-expert Inarti présenté 2 paragraphes plus haut; (cf. [Gar])

```

;principal
(define (resou cube-a-deplacer pile-consigne niveau-consigne)
  (define possible)

  (eff-lgn (+ 1 l-consigne))
  (eff-lgn l-consigne)
  (window-set-cursor! 'console (+ 1 l-consigne) 3)
  (write " But:cube=") (write cube-a-deplacer)
  (write " pile=")(write pile-consigne)
  (write " niveau=")(write niveau-consigne)
  (window-set-cursor! 'console (+ 1 l-consigne) 1)
  (if (verif-consigne cube-a-deplacer pile-consigne niveau-consigne)
    ;then
    (toutes-les-regles cube-a-deplacer pile-consigne niveau-consigne)
    ;else
    (write " Consigne erronée")
  )
  (eff-lgn (- l-consigne 1))
)

```

Fig.22 Extrait du module Lisp "Règles.s": définition de l'instruction principale "Résou[dre]"; (cf. [Gar])

Le système est disponible au laboratoire de l'HEIG-VD. L'expérience a montré que pour des applications complexes, telles la manipulation de cubes, le système est trop figé, et il est nécessaire de lui ajouter des instructions LISP, ce qui va à l'encontre de l'objectif de convivialité du système.

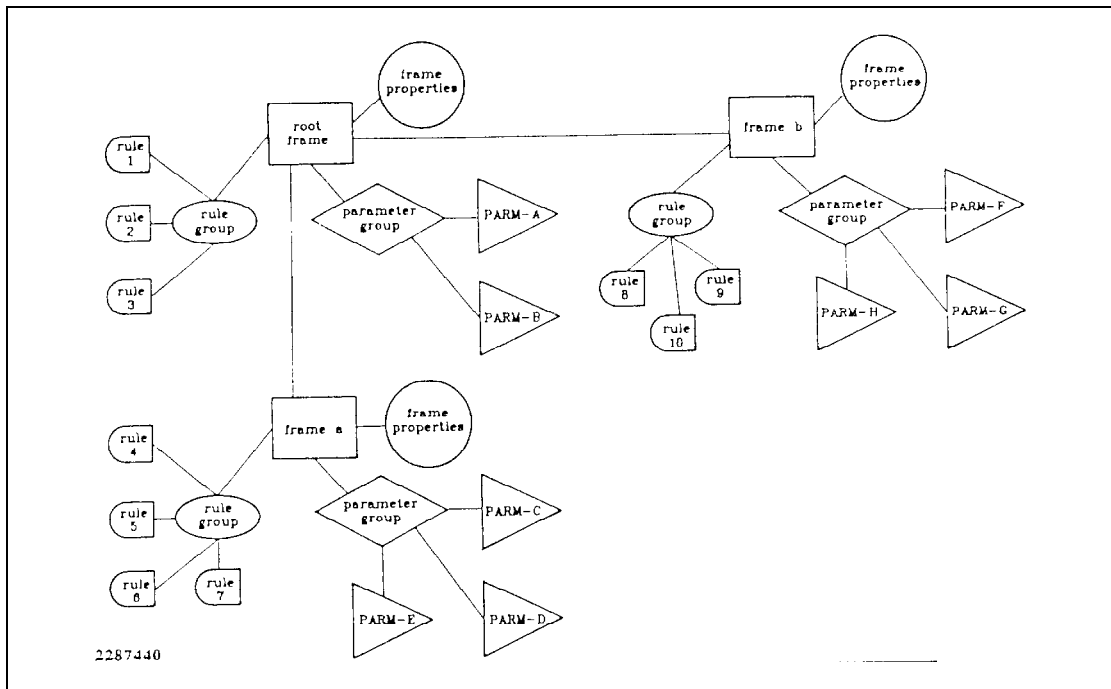


Fig.23 Le PGSE Personal Consultant Plus permet de structurer un problème sous la forme d'une arborescence de domaines et de sous-domaines (frames).

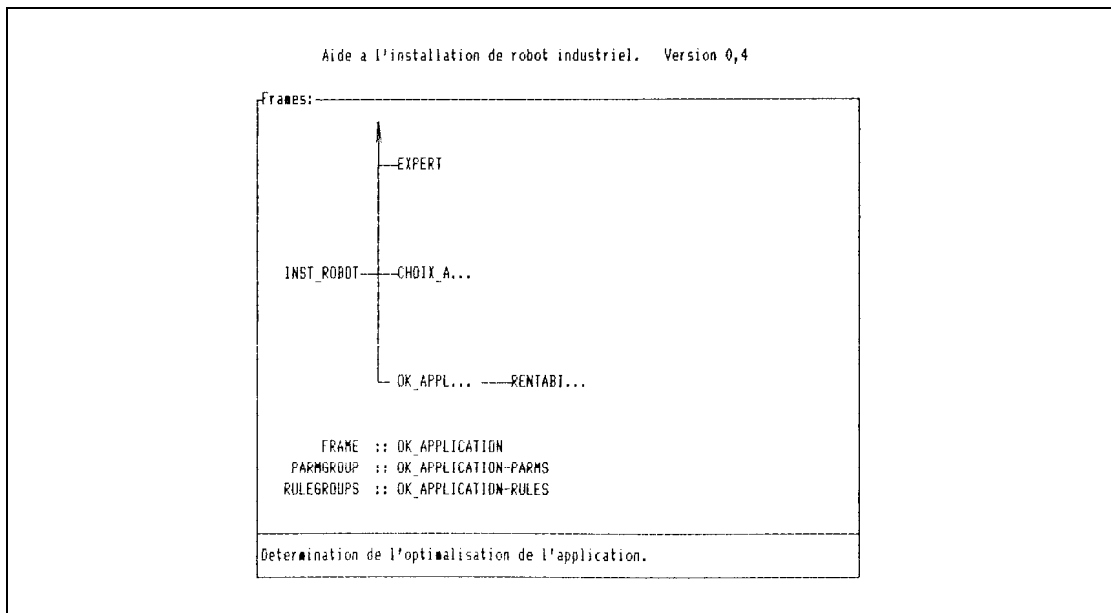


Fig.24 Exemple de structure d'un système d'aide à la décision écrit en PGSE [Per].

```

Rule Group [CHOIX_PERIROBOB-RULES]
RULE027 [CHOIX_PERIROBOB-RULES]
    If l'ordre des pièces en amont du robot is VRAC,
    Then it is definite(100%) that le positionnement des pièces.
    IF: ORDRE_PIECE = VRAC
    THEN: ORIENTATION_PIECE
RULE028 [CHOIX_PERIROBOB-RULES]
    If l'ordre des pièces en amont du robot is PALLETTE,
    Then it is definite(100%) that le positionnement des pièces is not true.
    IF: ORDRE_PIECE = PALLETTE
    THEN: ! ORIENTATION_PIECE
RULE029 [CHOIX_PERIROBOB-RULES]
    If 1) le positionnement des pièces, and
    2) l'état de surface des pièces constituant la production is COMPLEXE,
    Then it is definite(100%) that les éléments périphériques envisagés is CAMERA
    IF: ORIENTATION_PIECE AND SURFACE_PIECE = COMPLEXE
    THEN: ELEMENT_PERIROBOTIQUE = CAMERA
RULE034 [CHOIX_PERIROBOB-RULES]
    If 1) le positionnement des pièces, and
    2) la grandeur des pièces is PETITE, and
    3) la forme des pièces constituant la production is SIMPLE,
    Then it is definite(100%) that les éléments périphériques envisagés is BOL_VIBREUR
    IF: ORIENTATION_PIECE AND TAILLE_PIECE = PETITE AND FORME_PIECE=SIMPLE
    THEN: ELEMENT_PERIROBOTIQUE = BOL_VIBREUR
RULE035 [CHOIX_PERIROBOB-RULES]
    If 1) le positionnement des pièces, and
    2) la grandeur des pièces is GRANDE, and
    3) la forme des pièces constituant la production is SIMPLE,
    Then it is definite(100%) that les éléments périphériques envisagés is CHICANE
    IF: ORIENTATION_PIECE AND TAILLE_PIECE = GRANDE AND FORME_PIECE=SIMPLE
    THEN: ELEMENT_PERIROBOTIQUE = CHICANE
RULE035 [CHOIX_PERIROBOB-RULES] ....
    
```

Fig.25 Extraits d'un système d'aide à la décision écrit en PGSE [Per].

A.6 Problème de la manipulation de cubes

Le laboratoire de robotique de l'HEIG-VD a tenté de tirer parti des apports des systèmes-experts pour générer automatiquement des plans d'actions pour un robot. La tâche qui a servi de fil conducteur à travers ces divers essais, c'est celle de la manipulation de cubes. Bien qu'apparemment simple, cette tâche est représentative de la plupart des applications en manutention et en assemblage.

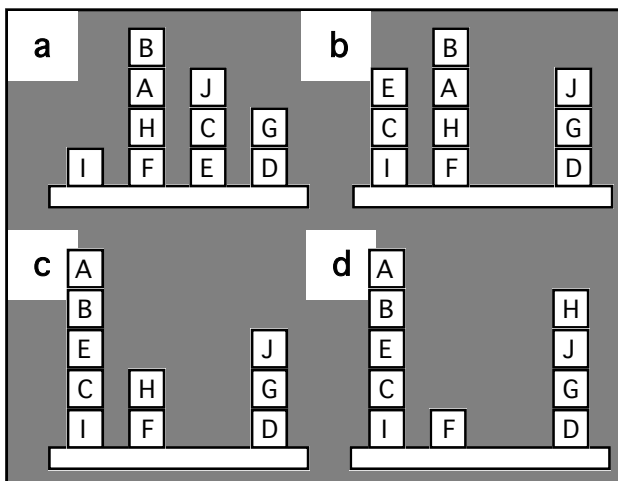


Fig.12 Quelques étapes du "raisonnement" d'un système-expert pour la manipulation de cubes. Des cubes, schématiquement représentés par des lettres, sont aléatoirement distribués sur quatre piles (a). Après réception de la commande "F31", c'est-à-dire "amener le cube F sur la 3e pile, au 1er niveau", le système déduit que la

3e pile doit être libérée (b), que les cubes empêchant l'accès au cube F doivent être déplacés (c), et qu'à ce moment-là seulement, le cube F peut être saisi et amené à destination (d).

Le nombre de mouvements possibles en fonction du nombre de composants et d'actions successives envisagées y croît selon les lois combinatoires, et il n'existe pratiquement pas d'algorithme permettant de trouver à chaque fois la meilleure solution. Pour ce type de problème, on estime aujourd'hui que l'I.A. et en particulier les systèmes à bases de connaissances offre la meilleure approche. Le nombre excessivement élevé d'alternatives à étudier se trouve sensiblement diminué par la prise en compte de "granules de connaissances" ("heuristiques", par exemple codées sous la forme de règles de production).

Plusieurs réalisations se sont faites, qui reflètent l'évolution générale qui s'est déroulée dans la branche (cf. point 4 ci-dessus). La première réalisation (logiciel Inarti) tourne sur Vax et s'est écrite dans un langage non-spécialisé, mais de haut niveau. Une deuxième réalisation s'est faite en utilisant le langage Prolog. Ensuite, c'est le progiciel de génération de systèmes experts Personal Consultant Plus qui a été sollicité. Cela a conduit à une application dans le domaine de l'installation de robot, mais pour la tâche des cubes, l'outil s'est avéré bien trop rigide. Comme ce dernier est écrit en Lisp, l'idée a fait son chemin de résoudre la tâche des cubes en Lisp, laissant éventuellement pour le futur l'intégration de ces deux réalisations (Lisp et Pers. Consult. +).

```

Commandes: <HLP>,<PRG>,<FIN>,<DBG>,<GO>,<Ctrl/C>
                                                    Stop
      H   B   I
      E   A   F
      C   D   G   J
      1   2   3   4

Donnez la consigne sous la forme: < nom_du_cube | pile | niveau>
Ex.: B32 ou FIN          Consigne: ?

```

Fig.13 Exemple de session interactive avec INARTI.

A.6.1 Inarti: Génération de plans pour l'assemblage de cubes

Inarti est le premier système à base de connaissance fait à l'HEIG-VD [Sol]. Destiné à résoudre le problème des cubes (voir § précédent), il a le mérite non seulement d'avoir été le premier développé mais encore de très bien marcher et d'apporter la solution, aujourd'hui encore, la plus conviviale. La discussion de cette réalisation se fait ici en plusieurs points, traitant successivement de l'interface utilisateur, du langage utilisé, des explications que le système donne sur le déroulement du raisonnement, ainsi que de la manière d'implémenter effectivement une règle de production.

Interface utilisateur

La fig. 13 représente schématiquement l'écran principal. On trouve en sommet d'écran un menu clignotant à 5 entrées, que l'option HLP (aide) détaille, un indicateur Stop/Run indiquant l'activité du système, les piles de cubes, apparaissant en gros caractères et vidéo inverse, ainsi que la zone de dialogue pour l'entrée des commandes. Dans l'un des modes, discuté plus bas, les explication défile dans le tiers inférieur de l'écran.

Langage utilisé

Le système est écrit en Vax Extended Basic. Comme tout langage de haut niveau non spécialisé (autres exemples: Pascal, Ada, Fortran, C...), VEB permet assez rapidement à l'utilisateur de programmer une application. Il est bien intégré au système d'exploitation, dispose de nombreuses fonctions d'entrées-sorties et possédait surtout pour nous, à l'époque, l'avantage d'être compatible avec le premier langage de programmation appris par les microtechniciens.

Le langage permet de résoudre notre application avec un ensemble relativement petit de règles de production (par contre certaines règles sont très puissantes, telle la règle "Tant que..." discutée plus loin).

Mais ici comme dans le cas de tout langage non-spécialisé, à la difficulté inévitable de conception d'un système cohérent de règles de production, il s'ajoute pour l'utilisateur la tâche de gestion de ces règles (écriture du moteur d'inférence).

Explication du raisonnement

Une option de déverminage ('DBG' dans le menu principal) permet de suivre le raisonnement du système lors de l'élaboration d'une solution. Dans ce mode là, le système affiche à l'écran les règles dont les conditions sont satisfaites, dans l'ordre de leur traitement. Certaines variables internes sont également affichées pour indiquer les variations progressives de la base de faits.

Dans l'état actuel du système, l'utilisateur gagne à garder simultanément un oeil sur la liste des règles (listing), et en particulier sur le commentaire qui en décrit le contenu, pour effectivement comprendre comment le système travaille.

Commentaire détaillé d'une règle.

Afin de mieux sentir comment le système travaille, regardons en détail le contenu de la règle 8 (fig. ci-dessous). Cette règle gère le cas où un cube à prendre n'est pas en sommet de pile. Elle redéfinit alors, le cas échéant, et de façon itérative, le cube à prendre comme étant celui du niveau immédiatement supérieur.

```

6400! REGLE 8 : Tant qu'il y a un cube au-dessus de celui a prendre      &
      !      alors c'est celui-la qui est a prendre.
6500IF CUB_A_PR$<>""      &
      THEN WHILE SEG$(PILE$(POSI_P%),POSI_N%+INC%,POSI_N%+INC%)<>""      &
      \      POSI_N%=POSI_N%+INC%      &
      \      CUB_A_PR$=SEG$(PILE$(POSI_P%),POSI_N%,POSI_N%)      &
      \      PRINT " R8: >>>";POSI_N% IF DBG%      &
      \      PRINT "   CUB_A_PR=";CUB_A_PR$ IF DBG%      &
      \      NEXT      &
      \      IF CUB_A_PR$<>CUBES$ AND DESTI_P%=PILE% AND DESTI_N%=NIVEAU% &
      THEN UNTIL ALEA%<>PILE% AND ALEA%<>POSI_P% AND
      \      \      LEN(PILE$(ALEA%))<MAXNIVEAU%      &
      \      \      ALEA%=INT(RND*4)      &
      \      \      NEXT      &
      \      \      DESTI_P%=ALEA%      &
      \      \      DESTI_N%=LEN(PILE$(DESTI_P%))+INC%      &
      \      \      END IF      &
      \      PRINT " R8b: DESTI_P%=";DESTI_P% IF DBG%      &
      \      PRINT "   DESTI_N%=";DESTI_N% IF DBG%      &
      \      PRINT "   CUB_A_PR=";CUB_A_PR$ IF DBG%      &
      \      END IF      &

```

Fig.14 Exemple de règles du système cognitif INARTI (implémentation Basic).

```

(* Règle 5: si il n'y a pas de cube a la position de consigne
   qu'il y en a un en dessous
   ou que c'est le premier de la pile alors
   c'est le cube de consigne qui est a prendre
*)

Cube_1 := Copy(Pile[PileCube-1],NivCube,1);
Cube_2 := Copy(Pile[PileCube-1],NivCube+Dec,1);

If (Cube_1 <> "") and (Cube_2 <> "") or (NivCube = 1) Then
Begin
  Cube_A_Pr := Cube;
  Pile_Dest:= PileCube;
  Niv_Dest:= NivCube;

End;

```

Fig. Exemple de règles du système cognitif INARTI (implémentation Pascal).

On constate tout d'abord que le programme représente les cubes par une variable de type caractère. La valeur de cette variable est familière à l'utilisateur, puisqu'elle se confond avec le nom du cube (A, B, etc.). La règle a pour première condition qu'il y ait un cube à prendre, et donc que la valeur de la variable CUBE_A_Prendre ne soit pas nulle.

La deuxième partie de la règle en est le coeur: c'est ce que le commentaire de la ligne 6400 décrit.

On y voit que les piles sont représentées par des éléments d'un tableau de chaînes de caractères (par ex. PILE\$(1), ou encore PILE\$(2)...). Ainsi le niveau d'un cube sur une pile devient ici la position d'un caractère dans une chaîne. La recherche d'un cube, ou son déplacement se réalise par des opérations sur les chaînes de caractères. Or pour ce type d'opération, il existe dans pratiquement tous les langages de haut niveau des instructions très efficaces (ainsi SEG\$ permet ici d'accéder à un cube quelconque sur une pile, et LEN indique la hauteur d'une pile).

On reconnaît au passage les affichages optionnels prévus pour le mode de déverminage (lignes terminées par 'IF DBG%').

La troisième partie de la règle concerne une situation singulière: la condition y est que le cube à prendre ne soit pas le cube de consigne (celui-ci est initialement désigné par l'utilisateur sur sa ligne commande), mais qu'il se trouve dans la même pile que celui-ci. Il s'agit donc d'un cube à dégager. On lui choisit de façon aléatoire une pile de dépôt, en s'assurant toutefois qu'il ne s'agisse ni de la pile courante, ni de la pile où le cube de destination devra être amené, et que de plus, la pile choisie ne soit pas pleine. Le niveau de dépôt du cube à prendre sera de 1 supérieur à la hauteur actuelle de la pile.

A.6.2 Implémentation Prolog

Prolog est un langage spécialement développé pour "programmer" des systèmes de règles (celles-ci sont appelées "clauses" dans le jargon de Prolog). C'est à l'aide d'un tel outil que le deuxième système à base de connaissance pour la manipulation de cubes, "Cube_pro" a été fait à l'HEIG-VD ([Sch])

Le moteur d'inférence fait partie du noyau Prolog, aussi suffit-il pour l'utilisateur de déclarer les règles ainsi que les grandeurs décrivant l'état courant du système (les "faits"). L'enchaînement des règles, c'est-à-dire l'ordre de leur évaluation respective est donc automatiquement assuré par le système. Il s'agit de "chaînage arrière" tel que défini plus haut dans ce chapitre (cf. §3).

Pour notre application, les cubes sont représentés par la chaîne de caractère "cube(...)" comprenant trois autres chaînes de caractères en paramètres internes: le nom du cube (a,b... ou "0" si c'est le sommet de la pile), et le numéro de la pile ainsi que le numéro du niveau où le cube est stocké.

```

cube(a,1,1) ;
cube(b,1,2) ;
cube(c,1,3) ;
cube(d,1,4) ;
cube(0,1,5) ;

```

Fig.15 Extrait de la base de faits: les cinq déclarations ci-dessus définissent le contenu initial de la pile No 1.

La figure ci-dessous présente un extrait typique d'une session de travail avec le système Cube_pro. L'utilisateur demande de déplacer le cube "a" sur la troisième pile au niveau No 4. Le système décrit alors l'état de la situation, puis indique les mouvements élémentaires successifs à accomplir.

```

/ ?- deplace(a,3,4).
d
c
b f
a e
1 2 3 4
prendre le cube d
mettre le cube sur la pile 3
c
b f
a e d
1 2 3 4
...
prendre le cube a
mettre le cube sur la pile 3
a
b
f c
e d
1 2 3 4
Solution 1 :
ok
next solution ? Y/N y
One solution.
Frame stack size : 1281
Number of created (pushed) frames : 1604

```

Fig.16 Extrait d'une session de travail avec Cube_pro. Les pointillés indiquent ici qu'une partie de la session est occultée.

Avant d'aller plus avant, il convient de mentionner deux choses fondamentales en prolog:

1. Tout symbole commençant par une majuscule est une variable.
2. Le système fait essentiellement des comparaisons systématiques de chaînes de caractères, dont l'issue n'est en principe que vrai ou faux.

La fig. ci-dessous illustre les deux points. L'état du système est celui de la fig. précédente. On a donc en particulier deux cubes dans la pile 2; le cube "f" y est posé sur le cube "e". Dans la commande donnée au système, on remarque deux éléments constants, donc écrits en minuscules: "cube(...)", et "2". Par ailleurs, nous avons deux éléments variables, "Cube", et "Niveau" (remarquons qu'ici seul l'état majuscule de la première lettre a d'importance, et que l'on aurait exactement le même résultat si l'on avait donné "CUZTRRCVG" et "KJHKJ" par exemple...).


```

/ ?- cube(Cube,2,Niveau).
Solution 1 :
  Cube=e,Niveau=1
next solution ? Y/N y
Solution 2 :
  Cube=f,Niveau=2
next solution ? Y/N y
Solution 3 :
  Cube=0,Niveau=3
next solution ? Y/N y
3 solutions.
Frame stack size : 1281
Number of created (pushed) frames : 1607

```

Fig.17 Le système trouve 2 cubes (e,f) et un sommet de pile ("0") sur la pile 2.

On constate que le système retrouve trois chaînes de caractères correspondant aux éléments constants imposés, et que dans ces trois cas, les parties variables valent respectivement (e,1), (f,2) et (0,3).

Prolog n'est pas un langage procédural, mais un langage déclaratif. Néanmoins, certaines déclarations sont en fait prétexte à démarrer des actions de nature essentiellement procédurales. Dans l'exemple ci-dessous, "is" est déclaratif (ceci de façon assez artificielle puisque son état ne varie pas et prend toujours la valeur "vrai"); elle est pourtant très utile parce qu'elle permet de calculer un paramètre (VAR) en fonction d'un autre (H).

En prolog, les règles s'écrivent dans l'ordre inverse de ce qui est naturel. Ainsi, les deux clauses de l'exemple ci-dessous représente, sous forme traditionnelle, la définition suivante:

```

SI
  - il existe un certain cube "C", dans n'importe quelle pile et à n'importe quel niveau,
  - et qu'au niveau immédiatement supérieur
  - il y a une fin de pile, et ceci dans la même pile
ALORS
  ce cube, C, est accessible.

```

```

/*Clause définissant un cube accessible donc au sommet d'une pile.*/
cube_accessible(C):-cube(C,P,H),is(VAR,H + 1),cube(0,P,VAR).

```

Fig.18 Exemple de clause en Prolog.

On pourrait aussi en français exprimer la clause sous la forme suivante:

```

POUR QUE
  un certain cube, C, soit accessible
IL FAUT QUE
  - il existe un certain cube "C", dans n'importe quelle pile et à n'importe quel niveau,
  - et qu'au niveau immédiatement supérieur
  - il y ait une fin de pile, et ceci dans la même pile

```

Voyons pour terminer une partie de la clause principale, c'est-à-dire de celle qui assure le déplacement des cubes (cf. fig. ci-dessous). Tout d'abord on constate que "déplace" se confond avec "déplacer", sauf qu'elle affiche sur 10 lignes l'état courant des piles.

```

/*Clause definissant l'initialisation du déplacement d'un cube. */
deplace (C,P,H) -> affiche (10), deplacer_cube(C,P,H);

/*Clauses definissant le déplacement du cube C dans la pile P a */
/*la hauteur H. */
deplacer_cube(C,P,H):-cube(C,P,H),!,print("Le cube est deja en place") ;
deplacer_cube(C,P,H):-cube(C,P,H1),cube(0,PT,HT),diff(P,PT),
    rendre_cube_accessible(C,PT,HT),prendre(C,P),
    mettre(C,PT),deplacer_cube(C,P,H),! ;
deplacer_cube(C,P,H):-rendre_cube_accessible(C,P,H),cube(C,P1,H1),
    rendre_position_accessible(P1,P,H),prendre(C,P1),
    mettre(C,P),!.
deplacer_cube(C,P,H):-prendre(C,P1),cube(0,P2,_),diff(P2,P),
    diff(P1,P2),mettre(C,P2),deplacer_cube(C,P,H),!;

```

Fig.19 Clauses principales de l'application Cube_pro

La première déclaration pour déplacer permet de détecter le cas où les paramètres donnés par l'utilisateur correspondent à une chaîne de caractère présente dans la base de fait. C'est donc que le cube choisi se trouve déjà dans la bonne pile, au niveau demandé. Remarquons ici le symbole "!". Prolog tente inlassablement de trouver des correspondances entre chaînes de la base de faits et toutes les variations possibles que les termes en majuscules autorisent. Le symbole en question interrompt cette recherche. En quelque sorte on pourrait dire que si ce symbole n'était pas là, le système irait vérifier qu'il n'existe pas d'autre cube avec le même nom, au même endroit.

En cas d'échec de la première déclaration de la clause, le système évalue la suivante, et ainsi de suite. Dans la deuxième déclaration, on gère le cas où le cube se trouve dans la bonne pile mais au mauvais niveau. La déclaration dit ce qui suit:

POUR

déplacer un certain cube, C, sur la pile P au niveau H

IL FAUT

- qu'il y ait ce cube "C" dans la bonne pile "P", et ceci à n'importe quel niveau,
- qu'il y ait un sommet de pile dans une pile quelconque...
- ... mais différente de celle où est C,
- y déplacer les cubes qui gênent l'accès à C
- puis prendre C et l'amener sur cette pile temporaire
- et rappeler la clause déplacer_cube (implémentation récursive!)

A.6.3 Implémentation Lisp

```
;definition de la classe pour le cube
;;;
;;;
;;; Class cube
;;;
;;;

(define-class cube
  (instvars (identite 'A)
            (NOM #A)
            (pile 1) ;repere horizontal, des lettres
            (niveau 1)) ;reperes vertical, des chiffres
  (options gettable-variables
           settable-variables
           inittable-variables))

(compile-class cube)
```

Fig.20 Extrait du module Lisp "Cubes.s": définition de la classe d'objets "cube", caractérisée par un nom de cube, ainsi d'une position en termes d'appartenance à une pile et un certain niveau de stockage; (cf. [Gar])