Conservatoire National des Arts et Métiers
Centre régional Rhône-Alpes
Centre d'Enseignement de GRENOBLE

MEMOIRE

présenté par Albino TEIXEIRA PEREIRA

en vue d'obtenir

Le Diplôme d'Ingénieur C.N.A.M.
en Informatique

Programmation bayésienne de comportements animaux sur un robot mobile

Soutenu le 28 Juin 2006

Jury

Président : Mme Véronique Donzeau-Gouge

Membres :

M. Eric Gressier

M. André Plisson

M. Jean-Pierre Giraudin

Mme Carla Koike

M. Pierre Bessière

Conservatoire National des Arts et Métiers

Centre régional Rhône-Alpes

Centre d'Enseignement de GRENOBLE

Mémoire présenté par

Albino TEIXEIRA PEREIRA

en vue d'obtenir

Le Diplôme d'Ingénieur C.N.A.M.
en Informatique

Programmation bayésienne de comportements animaux sur un robot mobile

---

Les travaux relatifs au présent mémoire ont été effectués, sous la direction de Carla Koike et Pierre Bessière, au sein de l'équipe de recherche e-Motion, dirigée par Christian Laugier.
e-Motion est un projet du laboratoire *GRAVIR*, UMR de l'INRIA, du CNRS, de L'INPG et de l'UJF. Il a pour principal axes de recherche la *modélisation multi-modale et incrémentale de l'espace et du mouvement*, la *planification de mouvement dans le monde physique* et l'*inférence probabiliste pour la décision*.

Remerciements

En préambule à ce mémoire, je souhaite adresser ici tous mes remerciements aux personnes qui m'ont apporté leur aide et sans qui, ce mémoire n'aurait jamais vu le jour.

Je remercie tout d'abord Carla Koike, qui a encadré ce stage, pour sa compétence, sa sensibilité et sa générosité. Merci pour l'aide et le temps précieux qu'elle a bien voulu me consacrer dans la rédaction de ce mémoire. *Muito Obrigado !*

Je souhaite remercier Monsieur Pierre Bessière, chargé de recherche au CNRS, pour ses conseils toujours pertinents et sa claire vision des choses.

Je remercie Monsieur André Plisson, directeur du centre d'enseignement de Grenoble, et toute son équipe pour leur aide précieuse tout au long de mon parcours avec le CNAM.

Mes remerciements à Monsieur Jean-Pierre Giraudin, professeur à l'Université Pierre-Mendès France de Grenoble et responsable du cycle ingénieur CNAM en informatique à Grenoble, pour son temps et ses critiques constructives dans la rédaction de ce mémoire.

Je suis très honoré par la présence au jury de Madame Véronique Donzeau-Gouge, professeur au CNAM de Paris et présidente de ce jury, et de Monsieur Eric Gressier, professeur au CNAM de Paris.

J'adresse également mes remerciements à Soraya Arias et Jean-François Cuniberto, membres du service SED, pour leurs compétences et leur engagement, et à Christophe Braillon pour sa disponibilité et ses conseils techniques précieux. Sans eux, l'expérimentation n'aurait pas abouti.

Un grand merci à tous les thésards et stagiaires de l'équipe e-Motion, aux grands maîtres de l'awele, à mes co-équipiers du tournoi de pétanque, sans eux, jamais l'ambiance n'aurait été la même.

Je suis particulièrement reconnaissant à tous mes proches et amis qui m'ont toujours soutenu et encouragé au cours de la réalisation de ce mémoire.

Enfin, plus que des remerciements pour celle qui a supporté sauts d'humeur, week-ends ensoleillés et pourtant studieux, et qui a su m'encourager dans les moments difficiles. J'embrasse tendrement mes enfants, Ruben et Elena, qui tout simplement par leur présence, me rendent heureux.

Table des matières

1 La programmation bayésienne **5**
- 1.1 Raisonnement probabiliste vs logique . . . 5
- 1.2 Définitions . . . 5
  - 1.2.1 Règles de calcul et théorème de Bayes . . . 5
  - 1.2.2 Règles dérivées . . . 6
- 1.3 Structure d'un programme bayésien . . . 6
  - 1.3.1 Spécification . . . 7
  - 1.3.2 Identification . . . 8
  - 1.3.3 Utilisation . . . 9
- 1.4 La fusion de données . . . 9
- 1.5 Le filtre bayésien . . . 12
  - 1.5.1 Variables . . . 13
  - 1.5.2 Décomposition . . . 13
  - 1.5.3 Formes paramétriques . . . 14
  - 1.5.4 Questions . . . 14
- 1.6 Contrôle sensori-moteur au moyen du filtre bayésien [Koi05] . . . 15
  - 1.6.1 Variables . . . 16
  - 1.6.2 Décomposition . . . 18
  - 1.6.3 Formes paramétriques . . . 18
  - 1.6.4 Questions . . . 19
- 1.7 Librairie *ProBt*© . . . 21
- 1.8 Synthèse . . . 24

2 Présentation de la plate-forme expérimentale **25**
- 2.1 Le robot BIBA . . . 25
  - 2.1.1 Le matériel . . . 25
  - 2.1.2 Le logiciel . . . 28
- 2.2 Extension des ressources . . . 32
- 2.3 Synthèse . . . 34

3 Comportement animal sur le robot BIBA **35**
- 3.1 Principes du comportement du robot . . . 35
- 3.2 Implémentation sur le robot BIBA . . . 35
- 3.3 Le programme bayésien . . . 37
  - 3.3.1 Le filtre élémentaire . . . 38
  - 3.3.2 Le filtre *maître* . . . 41

3.3.3 Le filtre *proie* . . . 45
3.3.4 Le filtre *prédateur* . . . 46
3.3.5 Le filtre *Route de fuite* . . . 48
3.4 Conclusion : algorithme d'utilisation des filtres . . . 48

4 Évitement d'obstacles réactif 51
4.1 Spécification . . . 51
4.1.1 Pré-traitement des données sensorielles . . . 51
4.1.2 Le sous-modèle proscriptif . . . 52
4.1.3 Le sous-modèle "suivi de consigne" . . . 53
4.1.4 Fusion des sous-modèles . . . 54
4.2 Identification des paramètres . . . 54
4.2.1 Le sous-modèle proscriptif . . . 54
4.2.2 Le sous-modèle "'suivi de consigne"' . . . 56
4.3 Utilisation . . . 58
4.4 Résultats de l'évitement d'obstacles réactif . . . 59
4.5 Conclusion . . . 60

5 Implémentation d'un comportement animal sur le robot BIBA 61
5.1 Un système distribué . . . 61
5.2 Les modules du système . . . 62
5.3 Outils utilisés . . . 62
5.3.1 Configuration du système . . . 64
5.4 Interface de bas niveau : *BBot* . . . 64
5.4.1 Spécifications . . . 64
5.4.2 Protocole de communication . . . 65
5.4.3 Mécanisme de gestion d'évènements . . . 66
5.4.4 Implémentation . . . 66
5.4.5 Traitement des commandes motrices . . . 67
5.4.6 Simulateur des données sensorielles . . . 69
5.5 Évitement d'obstacles : *BBotOBA* . . . 70
5.5.1 Spécifications . . . 70
5.5.2 Implémentation . . . 70
5.5.3 Création des zones . . . 71
5.5.4 Configuration du module . . . 73
5.6 Contrôleur de comportements : *BBehaviorCtrl* . . . 73
5.6.1 Spécifications . . . 73
5.6.2 Configuration du contrôleur . . . 74
5.6.3 Implémentation . . . 75
5.6.4 Le filtre élémentaire : *CFilter* . . . 75
5.6.5 Paramètres des modèles . . . 76
5.7 Capture et traitement d'image : *bbcam* et *camCGI* . . . 79
5.7.1 Spécifications . . . 79
5.7.2 Schéma d'architecture . . . 79
5.7.3 Exécution du système . . . 79
5.7.4 Le module *BBCam* . . . 81

5.7.5 Le module *CamCGI* 84

6 Résultats et validation **85**
6.1 Présentation des résultats 85
6.2 Analyse des résultats 87

7 Conclusion et perspectives **91**
7.1 Bilan 91
7.2 Les perspectives 92

Annexes **97**

A Guide utilisateur **97**
A.1 Description de l'expérience 97
A.2 Présentation du robot 98
A.2.1 Le robot Biba 98
A.2.2 Distribution des tâches 98
A.3 Démarrer une démonstration 99
A.3.1 Le démarrage du robot 100
A.3.2 Exécution des modules 102
A.4 Arrêter la démonstration 106

Table des figures

1.1 Structure d'un programme bayésien 7
1.2 Vue de dessus du robot et de ses capteurs infrarouge [Leb99] 10
1.3 Fusion bayésienne : comportement phototaxique 11
1.4 Fusion des données de huit capteurs infrarouge pour estimer la direction d'une source lumineuse 12
1.5 Le filtre bayésien vu comme une boucle prédiction-estimation 13
1.6 Filtre bayésien : le programme 15
1.7 Boucle principale du filtre étendu [Koi05] 17
1.8 Filtre bayésien étendu : le programme 21
1.9 Exemple simple de programme bayésien 22

2.1 Le robot Biba développé par *Bluebotics*(www.bluebotics.com) 26
2.2 Les calculateurs embarqués 28
2.3 Utilisation de l'interface de bas niveau via un navigateur internet . . 30
2.4 Robot : Le pointeur laser et sa protection anti-éblouissement 32
2.5 Bibabot et le poseur de jetons 33
2.6 Bibabot et son contacteur de recharge 33

3.1 À gauche, le robot face au prédateur et au maître, pourchassant la proie à droite 36
3.2 Robot capturant sa proie plutôt souriante 37
3.3 Distribution des tâches sur les calculateurs 38
3.4 Z_{pan} donne l'orientation de la caméra 39
3.5 Convention pour estimer la direction des objets 40

4.1 Évitement d'obstacles : variables d'observations 52
4.2 Évitement d'obstacles : sous-modèle bayésien proscriptif 53
4.3 Évitement d'obstacles : fusion des sous-modèles 54
4.4 Évitement d'obstacles : les seuils de contraintes 55
4.5 Modèle suivi de consigne : identification des paramètres 58

5.1 Distribution des tâches 62
5.2 Diagramme de l'architecture 63
5.3 La classe d'interface pour les fichiers de configuration 64
5.4 Implémentation de la classe générique *CProtocole* 65
5.5 Diagramme de classes du mécanisme d'événements 67
5.6 Diagramme de classes du module *BBot* 68

5.7 Implémentation de la classe générique *CModule* 69
5.8 Traitement des commandes motrices : à gauche, limitation de la force centrifuge ; à droite, limitation des accélérations 70
5.9 Diagramme de classes du module d'évitement d'obstacles 71
5.10 Diagramme de classes du contrôleur de comportements 75
5.11 Schéma d'architecture du système d'acquisition et de traitement d'image 80
5.12 Détection des trois teeshirts 82

6.1 Comportement : chasse. Description des photos : (a) Le robot se déplace librement en évitant les obstacles ; (b) Il détecte une proie sur son côté gauche ; (c) Le robot se déplace en direction de la proie ; (d) Il approche la proie autant que possible ; (e) Le robot s'arrête pour capturer la proie ; (f) La proie est capturée avec le pointeur laser. . . 86
6.2 Comportement : obéissance avec évitement d'obstacles. Description des photos : (a) Le robot détecte le maître ; (b) Il suit son maître ; (c) Un obstacle est sur la trajectoire ; (d) Le robot change de direction pour éviter l'obstacle ; (e) Il contourne l'obstacle tout en « regardant » son maître ; (f) Le robot s'approche de son maître. 87
6.3 Comportement : chasse puis réactivité face au prédateur.Description des photos : (a) Le robot détecte une proie ; (b) Il poursuit la proie ; (c) Un prédateur apparaît ; (d) Le robot s'arrête de chasser et cherche une route de fuite ; (e) Le robot s'échappe ; (f) Il se cache du prédateur. 88
6.4 Comportement : coordination. Description des photos : (a), (b) Le robot se déplace librement ; (c) Il détecte assez loin un prédateur ; (c), (d) Il reste immobile ; (e) Il détecte son maître qui approche ; (f), (g), (h) Le robot suit son maître ; (i) Même en présence du prédateur, le robot reste à côté de son maître. 89

A.1 Le robot Biba développé par *Bluebotics*(www.bluebotics.com) 99
A.2 Structure des calculateurs embarqués 100
A.3 Le PC portable : connexion au robot 101
A.4 Ne pas mélanger les sets de batteries 102
A.5 Structure logicielle 103

Liste des tableaux

2.1 Fiche technique de la caméra : ORANGE MICRO - IBOT 26
2.2 Principales commandes du module *PPCtoPC* 31
2.3 Principales commandes du module *Preferences* 31
2.4 Accessoires ajoutés au robot 32

3.1 Modèle dynamique du maître – *présence* 42
3.2 Modèle dynamique du maître – *distance* 42
3.3 Modèle dynamique du maître 43
3.4 Modèle capteur du maître – *présence* 43
3.5 Modèle capteur du maître – *distance* 44
3.6 Modèle capteur du maître – *direction* 44
3.7 Modèle direct de comportement du maître 45
3.8 Modèle direct moteur du maître – *vitesse de rotation* 45
3.9 Modèle direct moteur du maître – *vitesse de translation* 45
3.10 Modèle capteur du prédateur – *Présence* 47
3.11 Modèle de comportement du prédateur 48
3.12 Modèle de comportement du filtre route de fuite 49

4.1 Évitement d'obstacles : f(Vrot, Distance) 56
4.2 Évitement d'obstacles : f(Vtrans, Distance) 57
4.3 Évitement d'obstacles : situation expérimentale 59
4.4 Évitement d'obstacles : résultat de la fusion des sous-modèles 60

5.1 Accessoires ajoutés au robot 64
5.2 Calcul des seuils de contraintes 72
5.3 Calcul des seuils de contraintes 73
5.4 Fichier de configuration du contrôleur de comportements 74
5.5 Paramétrage des modèles en utilisant un fichier texte 77
5.6 Paramètres des modèles : fichier XML 78
5.7 Paramètres de la ligne de commande de *Blinky*. 80
5.8 Fichier de configuration du système d'acquisition et de traitement vidéo 83
5.9 Format de la page HTML 84

Introduction

Des réalisations comme *L'Ecrivain* de Pierre Jacquet-Droz (1774), un automate qui simulait l'écriture manuscrite, ou encore le *Canard* de Vaucanson (1738) représentent la volonté de l'homme, au fil des âges, d'essayer de créer des machines capables de mimer le vivant.

Aujourd'hui, mimer le vivant est toujours d'actualité. Outre la robotique ludique avec des robots capables de mimer des comportements ou des émotions (la peluche « Furby », le chien « Aibo », etc.), les sciences du comportement animal et humain, l'éthologie, utilisent les robots comme simulateur pour comprendre et tester leurs hypothèses. D'autre part, la robotique fait face à des problèmes auxquels la nature a trouvé des solutions.

L'incertitude et l'imprécision sont des notions auxquelles se confrontent les robots, mais également les êtres vivants. Ils évoluent dans un environnement incertain : les phénomènes physiques ou les mouvements imprévisibles d'autres êtres vivants. L'information obtenue des capteurs peut être bruitée ou manquante et les commandes motrices imprécises. Dans le cas du robot, les calculs peuvent être approximatifs. Cependant, les humains et les animaux ont su s'adapter à ces problèmes.

L'équipe e-Motion du laboratoire GRAVIR (GRAphics, VIsion and Robotics) a fait de cette problématique un de ses principaux axes de recherche. Elle utilise la méthodologie et le formalisme de la « Programmation Bayésienne » pour développer des artefacts qui soient capables de traiter des informations incomplètes et incertaines.

Dans le cadre d'un projet européen (2001-2005) nommé BIBA (Bayésien Inspired Brain and Artefacts), Carla KOIKE, membre de l'équipe e-Motion, a travaillé sur une méthode générique, pour l'implémentation de comportement sur un robot. Le canevas proposé est entièrement basé sur l'approche bayésienne, plus précisément sur le filtre bayésien, qui traite naturellement l'incomplétude.

Notre contribution

Le sujet de ce stage fut d'utiliser le canevas cité ci-dessus, pour implémenter un comportement animal simple sur un robot, l'objectif étant de montrer que la méthode n'est pas simplement théorique, mais qu'elle peut être utilisée en pratique.

Le robot doit pouvoir se déplacer librement dans un environnement fermé, tout en évitant les obstacles. Le robot ne possède pas la carte des lieux, en l'occurrence les bâtiments de l'INRIA. Il interagit avec quatre centres d'intérêts : un prédateur,

une proie, son maître et une route de fuite.

Face à un prédateur, il tente de se protéger. Près de son ennemi, il cherche à s'échapper, si une route de fuite est trouvée. Éloigné de celui-ci, il reste immobile pour ne pas être vu.

En présence du maître, le robot se sent en sécurité. Il ne fuit pas les éventuels prédateurs. Le robot suit son maître tant qu'il le perçoit. Si le robot détecte une proie, il la prend en chasse, et lorsqu'elle est assez proche, il la « capture ».

La librairie *ProBt*©

Pour automatiser l'inférence et le calcul bayésien, l'équipe e-Motion a mis au point une librairie, appelée *ProBt*©, qui permet d'implémenter les programmes bayésiens en langage $C++$.

Cette librairie est commercialisée par la société ProBayes[1]. Elle est également disponible gratuitement pour des utilisations destinées à la recherche et à l'enseignement.

Nous utiliserons cette librairie pour implémenter les programmes bayésiens présentés tout au long de ce mémoire.

Plan de lecture

Notre document traite de l'implémentation d'un comportement animal sur un robot mobile, au travers de 8 chapitres.

Nous commencerons par présenter la Programmation Bayésienne comme une méthode et un formalisme pour le développement de logiciels qui soient capables de traiter des informations incomplètes et incertaines. Nous aborderons les techniques et les outils bayésiens utilisés dans nos travaux : le filtre bayésien, la fusion de données, etc. Nous présenterons alors le canevas proposé par [Koi05] pour l'implémentation d'un comportement sur un système sensori-moteur. Enfin, nous ferons une brève introduction de la librairie *ProBt*©.

Le chapitre 3 présente la plate-forme expérimentale, le robot BIBA, avec ses aspects matériels (capteurs et actionneurs) et logiciels (contrôleur bas niveau embarqué).

Le chapitre 4 détaille l'implémentation du comportement spécifié sur le robot en utilisant le canevas proposé.

Pour garantir la sécurité du robot et de son environnement pendant ses déplacements, nous serons nécessairement confrontés au problème d'évitement d'obstacles. Le chapitre 5 présente la méthode utilisée, dérivée de [KPBM03].

Le chapitre 6 détaille le développement du contrôleur de comportements, le module d'évitements d'obstacles, ainsi que les librairies de prétraitements des commandes motrices et des données sensorielles, dont la vision.

Les résultats de l'exécution de l'application ainsi obtenue sont présentés au chapitre 7. Ces résultats sont comparés au comportement spécifié, ainsi que les méthodes de validation.

[1] http ://www.probayes.com

Enfin, nous conclurons avec une synthèse du travail réalisé, les perspectives de cette expérimentation et un bilan personnel sur le stage au sein d'un laboratoire de recherche.

Chapitre 1

La programmation bayésienne

Nous présentons dans ce chapitre la « Programmation Bayésienne », un outil formel pour le développement de logiciels qui soit capable de traiter des informations incomplètes et incertaines.

1.1 Raisonnement probabiliste vs logique

La théorie du raisonnement probabiliste, aussi nommée « Probability as Logic » (PaL), a été proposée par le physicien américain E.T. Jaynes en 1994, et elle est présente comme une extension de la logique booléenne.

Une proposition logique est un énoncé qui est soit vrai soit faux. Cependant, dans bien des cas, les informations disponibles ne permettent pas de trancher avec une certitude absolue. Nous définissons alors la « plausibilité » d'une proposition comme le degré de certitude que nous avons dans sa véracité. Cette plausibilité dépend évidemment des connaissances mises en jeu pour l'établir.

On peut donc voir l'approche logique comme un cas particulier de l'approche probabiliste. En effet, en limitant la probabilité des propositions à 0 ou 1, le problème est réduit à un raisonnement logique.

1.2 Définitions

1.2.1 Règles de calcul et théorème de Bayes

Nous résumons ici les deux règles fondamentales nécessaires à l'inférence bayésienne. Étant donné les propositions logiques A et B, ces deux règles sont la règle du produit et la règle de normalisation.

a. La règle du produit

La règle du produit, également appelée *théorème de Bayes*, permet d'exprimer la probabilité de la conjonction de deux termes A et B comme le produit de deux probabilités élémentaires :

$$P(AB|C) = P(A|C)P(B|AC) = P(B|C)P(A|BC)$$

Les variables recherchées, en l'occurrence A et B, doivent apparaître une et une seule fois à gauche dans la décomposition en probabilités élémentaires. Les deux formes possibles proviennent de la commutativité logique.

b. La règle de normalisation

La règle de normalisation exprime le fait que la somme des probabilités de tous les cas possibles d'un terme vaut 1.

$$P(A|C) + P(\neg A|C) = 1$$

1.2.2 Règles dérivées

a. Marginalisation

De ces deux règles fondamentales, peut être dérivée la *règle de marginalisation*, utilisée pour la simplification d'expression :

$$\sum_A P(AB|C) = P(B|C)$$

b. Formule de Bayes

Une réécriture directe de la règle du produit permet d'obtenir la formule dite de Bayes :

$$P(X|Y) = \frac{P(XY)}{P(Y)} = \frac{P(X)P(Y|X)}{P(Y)} = \frac{P(X)P(Y|X)}{\sum_X P(X)P(Y|X)}$$

1.3 Structure d'un programme bayésien

Dans cette section, notre objectif est de présenter brièvement les principes de la méthode appelée « programmation bayésienne » basée sur un objet formel, la *description* [Leb99]. L'élaboration et l'utilisation d'un programme bayésien se décompose en trois phases :

- la spécification des connaissances préalables,
- l'identification des valeurs des paramètres des distributions de probabilités,
- l'utilisation de la description.

Nous allons maintenant définir le concept de description et décrire plus précisément chacune de ces trois étapes. La figure 1.1 illustre la structure générale que nous utiliserons pour définir un programme bayésien.

Description

La description est la brique de base de la programmation bayésienne. Elle est dénotée formellement par une distribution de probabilités conjointe définie à partir

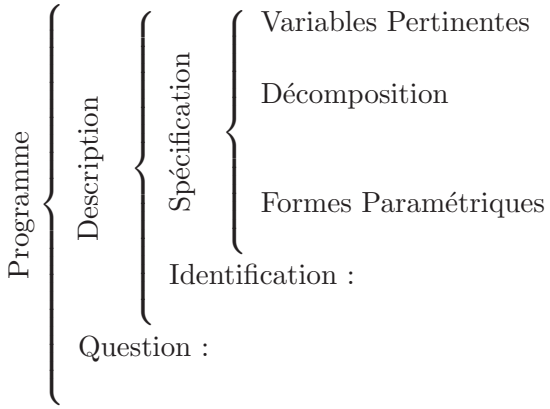


Fig. 1.1 : Structure d'un programme bayésien

de connaissances préalables du problème (notées C) et d'un ensemble de données expérimentales (notées D) :

$$P(V1 \ldots Vn \mid D\ C),$$

V1...Vn	:	ensemble des variables,
D	:	données expérimentales,
C	:	connaissances préalables.

1.3.1 Spécification

La phase de spécification est la partie la plus délicate du travail du programmeur. Au cours de cette phase, il doit énoncer explicitement les connaissances dont il est à l'origine et celles qui résultent d'un processus adaptatif dépendant d'un jeu particulier de données expérimentales. Ces connaissances se subdivisent en trois : le choix des variables pertinentes, l'expression des dépendances entre les variables retenues sous la forme d'un produit de distributions élémentaires, et enfin la forme paramétrique associée à chacune de ces distributions.

a. Connaissances préalables structurelles : le choix des variables pertinentes

Nous appelons *connaissances préalables structurelles* les connaissances permettant de définir l'ensemble des variables $V_1, \ldots, V_n$ pour la description, et de spécifier pour chacune d'elles son domaine de variation D_{V_i} et le nombre K_{V_i} d'états possibles. Toutes les autres variables sont ainsi supposées non pertinentes pour le problème considéré.

En robotique, ces variables sont naturellement classées en trois sous-ensembles :

- les variables sensorielles,
- les variables motrices,
- et enfin les variables internes, qui permettent de coder les états internes du robot.

b. Connaissances préalables de dépendance : le choix d'une décomposition de la distribution conjointe
Comme énoncé précédemment, la description sur les variables $V_1, \ldots, V_n$ a pour but la définition de la distribution conjointe $P(V_1 \ldots V_n | DC)$. Cette forme mathématique est une distribution de probabilités sur n dimensions. La règle du produit 1.2.1 nous permet de décomposer cette expression, en l'exprimant sous forme de produit de distributions.
Prenons en exemple une distribution conjointe $P(X\ Y\ Z)$ de 3 variables X, Y et Z. L'application de la règle du produit permet d'écrire :

$$P(X\ Y\ Z) = P(Z)P(Y \mid Z)P(X \mid Y\ Z).$$

Cette seconde étape de la spécification permet également d'exprimer les relations de dépendance, ou d'indépendance, entre les variables. Ces indépendances permettent de réduire fortement les dimensions des termes apparaissant dans la décomposition.
Si l'on suppose dans notre exemple que les variables X et Y sont indépendantes sachant la valeur de Z, on aura :

$$P(X\ Y\ Z) = P(Z)P(Y \mid Z)P(X \mid Z).$$

c. Connaissances préalables d'observation : le choix des formes paramétriques
Il faut maintenant associer à chacun des termes apparaissant dans la décomposition choisie à l'étape précédente une forme paramétrique. Par ces choix, nous allons fournir des connaissances a priori sur les valeurs des distributions de probabilités et la manière dont ces valeurs seront modifiées par l'expérience. Ce dernier point est composé d'un ensemble de valeurs initiales pour les paramètres, et d'un mécanisme de mise à jour au vu de données expérimentales (ce mécanisme peut éventuellement être vide si l'on veut figer la distribution lors de la présente phase de spécification). Une description dans laquelle tous les termes sont ainsi fixés est appelée *spécification (description) a priori*.

Les formes paramétriques sont en général des lois de probabilités classiques, par exemple des lois uniformes ou des lois normales. Une *question* à une autre description peut également être utilisée comme forme paramétrique, à la manière d'un sous-programme probabiliste.

1.3.2 Identification

Les formes paramétriques peuvent contenir des paramètres libres, comme les moyennes ou écarts types de distributions gaussiennes. Il est nécessaire de fixer les valeurs numériques de ces paramètres pour achever notre description. Ces valeurs peuvent, soit être obtenues par un processus d'apprentissage, soit être fixées a priori par le programmeur.

1.3.3 Utilisation

Au terme des phases de spécification et d'identification, nous disposons d'une description complètement définie. La phase d'utilisation va consister à mettre en oeuvre les descriptions par le biais de questions probabilistes.

a. Question
Poser une question consiste à chercher la distribution de probabilités d'un certain nombre de variables ϵ_q de la description, connaissant les valeurs d'autres variables ϵ_c, et éventuellement ignorant les valeurs d'un troisième groupe de variables ϵ_i. Une question probabiliste est donc n'importe quelle expression de la forme :

$$P(V_k \dots V_l | v_m \dots v_n), \tag{1.1}$$

où $\epsilon_q = V_k, \dots, V_l \neq 0, \epsilon_c = V_m, \dots, V_n$, et $\epsilon_i = V_o, \dots, V_p$ est l'ensemble des variables n'apparaissant ni dans ϵ_q, ni dans ϵ_c. Ces trois ensembles doivent bien sûr former une partition de l'ensemble des variables considérées pour que la question ait un sens.

b. Décision
Le résultat de l'inférence fourni une distribution de probabilités sur les variables recherchées. Cette distribution de probabilité résume les connaissances préalables du programmeur sur le problème et les informations apportées par des observations capteur par exemple.
Dans le cadre de la robotique, cette distribution de probabilités porte typiquement sur les variables motrices. Afin de contrôler le robot, il convient de choisir une valeur pour ces variables. Il s'agit donc d'un problème de décision. De nombreuses stratégies sont ici imaginables, mais les plus simples restent de choisir la valeur correspondant au maximum de probabilité, ou encore de tirer les valeurs aléatoirement selon la distribution obtenue.

1.4 La fusion de données

Très souvent l'analyse d'une information isolée ne permet pas de tirer des conclusions fiables. Par contre le recoupement et le croisement avec d'autres données permet d'en dégager une observation plus précise. Ceci est particulièrement vrai en robotique pour le traitement des données sensorielles. Ces dernières peuvent être plus ou moins précises du fait de leur dépendance des caractéristiques du système sensoriel, telles que la technologie des capteurs, leur précision limitée, ainsi que leur emplacement sur le robot.

D'où la nécessité d'un mécanisme bayésien de « recoupement et croisement » de données. Les travaux de [Leb99] et [Cou03], pour n'en citer que quelques uns, montrent que les techniques de fusion bayésienne permettent de gérer ces incertitudes de façon tout à fait satisfaisante.

La fusion bayésienne peut se résumer à la multiplication des distributions de probabilités issues des modèles sensoriels associés à chaque capteur. La distribution ainsi obtenue met en évidence l'estimation la plus probable et atténue le niveau des autres valeurs.

En posant l'hypothèse que les observations Z_j sont indépendantes les unes des autres nous pouvons écrire la fusion de modèles afin d'obtenir l'état S sous la forme :

$$\begin{aligned} & P(SZ_j|\pi) \\ & \quad = \quad P(S|\pi) \prod_{j=1}^{n} [P(Z_j|S\pi)] \end{aligned}$$

On retrouve dans cette décomposition les n modèles capteurs $P(Z_j|S\pi)$ fusionnés par le produit $\prod_{j=1}^{n} [P(Z_j|S\pi)]$.

Exemple d'utilisation

Illustrons maintenant l'utilisation de la fusion bayésienne au travers des travaux de [Leb99] sur le robot khépéra plus précisément l'implémentation du comportement de phototaxie. Le robot est équipé de huit capteurs infrarouges, placés de façon à couvrir les 360 degrés autour du robot comme illustré par la figure 1.2.

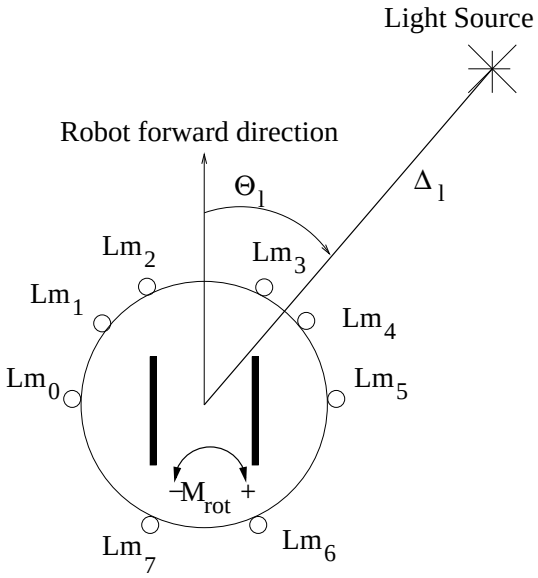


Fig. 1.2 : Vue de dessus du robot et de ses capteurs infrarouge [Leb99]

La direction de la source lumineuse est indiquée par la variable d'état θ. Elle peut prendre l'une des 36 valeurs de $-180°$ à $+170°$ par pas de $10°$. La valeur des variables d'observations, notées $lm0 - lm7$, peut varier de 0 (forte luminosité) à 511 (faible luminosité). Le comportement phototaxique peut alors être implémenté selon le programme bayésien décrit en figure 1.3.

Le premier terme de la décomposition, $P(\theta|\pi)$, exprime les connaissances a priori de la direction de la source lumineuse. Cette source lumineuse pouvant être placée n'importe où autour du robot, la forme paramétrique du terme est uniforme.

Le second terme est la fusion des modèles capteurs. Chaque modèle décrit la distribution de probabilités sur les mesures par rapport à une direction donnée. Le modèle est construit d'après la documentation technique du constructeur du capteur.

Programme
- Description
  - Spécification
    - Variables Pertinentes
      - θ : Direction de la source lumineuse ;
      - $lm0, \ldots, lm7$: intensité lumineuse mesurée.
    - Décomposition
      - $P(\theta\ lm0 \ldots lm7)$
      - $= P(\theta|\pi)$
      - $\times \prod_{i=0}^{7} P(lmi \mid \theta)$
    - Formes Paramétriques
      - $P(\theta|\pi)$: Uniforme ;
      - $P(lmi \mid \theta)$: Question aux huit sous-modèles capteurs.
  - Identification :
    - *A priori*
- Question :
  - $P(\theta \mid lm0 \ldots lm7)$

Fig. 1.3 : Fusion bayésienne : comportement phototaxique

Résultats

La figure 1.4 illustre les résultats de la simulation effectuée sur le programme ci-dessus. La source lumineuse est positionnée devant le robot, légèrement sur la droite.

Les distributions de chaque modèle sont positionnées suivant l'orientation des capteurs sur le robot. Nous pouvons noter que les distributions sur lm_2, lm_3 et lm_4 indiquent que la source lumineuse se trouve dans leur champ de vision, avec une certaine incertitude cependant. Les autres modèles, quand à eux, indiquent les directions dans lesquelles la source lumineuse ne peut pas se trouver, car elle est en dehors de leur champ de vision.

Au centre, la distribution de probabilités sur θ est issue de la fusion des huit modèles sensoriels. La direction réelle de la source lumineuse est mise en évidence avec certitude. Il est intéressant de noter que, de la même façon que la redondance d'informations augmente la précision des observations, l'absence de détection d'un capteur permet également d'accroître la certitude de la distribution de probabilité issue de la fusion.

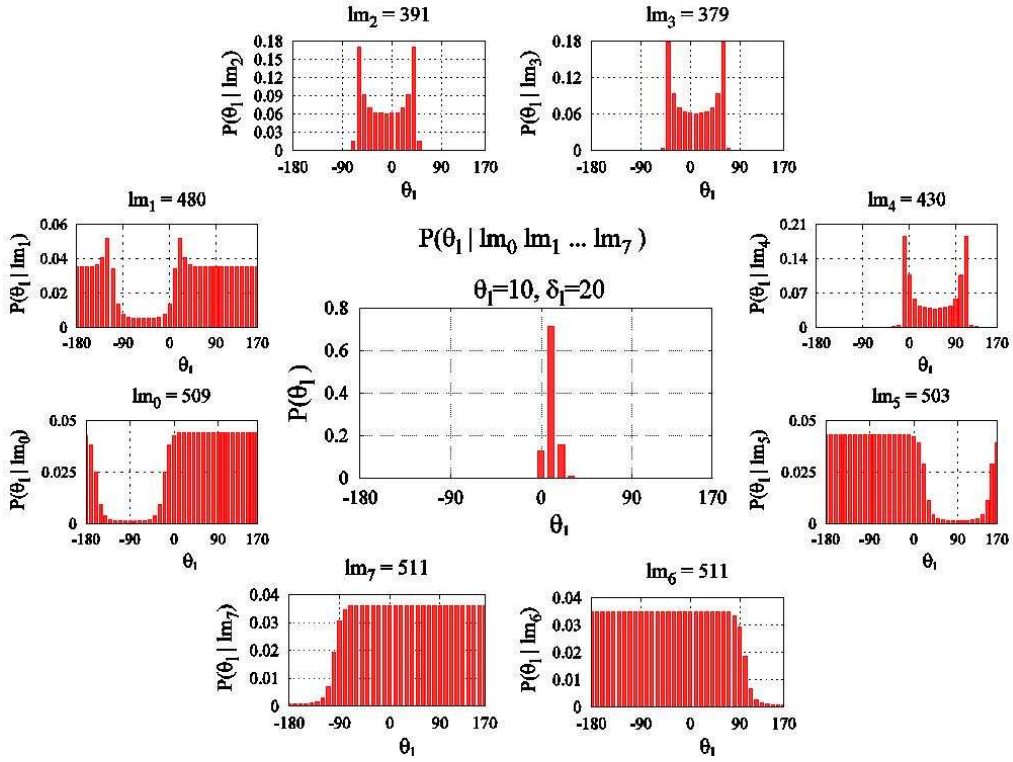


Fig. 1.4 : Fusion des données de huit capteurs infrarouge pour estimer la direction d'une source lumineuse

Conclusion

La fusion de capteurs a plusieurs avantages. Elle permet d'augmenter la précision des mesures, d'améliorer l'estimation des observations, de rendre le système sensoriel plus robuste aux perturbations.
Le modèle capteur faisant abstraction du type de capteur, il est ainsi possible de fusionner des modèles capteurs de technologies différentes (ie. infrarouge, ultrasons, laser ...) pour accroître la fiabilité d'une même observation.

1.5 Le filtre bayésien

La notion de filtre bayésien est le fruit de recherches sur l'estimation de l'état d'un système dynamique à partir d'observations bruitées, voir même manquantes.

Par exemple, un système observe, par le biais d'une caméra, un véhicule en mouvement. Il connaît sa vitesse, la taille du bâtiment et à quel moment celui-ci est passé derrière un bâtiment. Sans nouvelle observation, le système est capable de prédire la position du véhicule et estimer le moment où il sera de nouveau visible. L'état du système est alors mis à jour dès que le véhicule sera de nouveau dans le champ de vision de la caméra.

Dans cette approche, le système dynamique dont on veut estimer l'état, est supposé être un système de Markov d'ordre 1. Ainsi entre deux pas de temps, l'évolution de l'état du système, noté S, peut être modélisé par la distribution de probabilités

définie par l'équation 1.2. L'historique de l'état du système est résumé dans S^{k-1}.

$$P(S^k \mid S^{k-1}) \tag{1.2}$$

Pour estimer l'état du système à partir d'une observation à l'instant k, noté Z^k, il est également nécessaire de définir un modèle capteur :

$$P(Z^k \mid S^k) \tag{1.3}$$

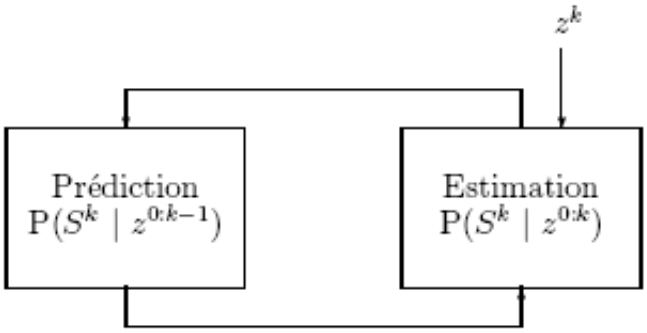


Fig. 1.5 : Le filtre bayésien vu comme une boucle prédiction-estimation

Le filtre bayésien peut être schématisé par la figure 1.5 tiré de [Cou03]. À chaque instant k la phase de *prédiction* extrapole l'état du système à l'instant $k-1$. La phase d'estimation met à jour cette prédiction avec la nouvelle observation capteur Z^k.

Nous allons maintenant présenter le programme bayésien du filtre.

1.5.1 Variables

Nous représentons l'état du système à un instant $k, k = 0 \ldots N$ par une variable S^k. L'observation capteur à chaque instant k est représentée par la variable Z^k. Afin de faciliter l'écriture et la lecture, nous notons $Z^{0:k}$ la conjonction des variables d'observation entre l'instant initial et l'instant k : $Z^0 \ldots Z^k$. De la même manière, nous notons $S^{0:k}$ la conjonction $S^0 \ldots S^k$. Ces variables définissent la distribution de probabilités conjointe :

$$P(Z^0 \ldots Z^N \ S^0 \ldots S^N) = P(S^{0:N} \ Z^{0:N}). \tag{1.4}$$

1.5.2 Décomposition

En appliquant la règle du produit à la distribution de probabilités conjointe 1.4, on obtient la décomposition :

$$P(S^{0:N}\ Z^{0:N}) = P(S^{0:N-1}\ Z^{0:N-1})P(S^N \mid S^{0:N-1}\ Z^{0:N-1})P(Z^N \mid S^{0:N}\ Z^{0:N-1}). \tag{1.5}$$

On retrouve dans cette décomposition un modèle récursif $P(S^{0:N-1}\ Z^{0:N-1})$, le modèle dynamique $P(S^N \mid S^{0:N-1}\ Z^{0:N-1})$ et le modèle capteur $P(Z^N \mid S^{0:N}\ Z^{0:N-1})$.

L'historique de l'état du système étant résumé dans l'instant $N-1$ (système de Markov d'ordre 1) le modèle dynamique peut se réécrire :

$$P(S^N \mid S^{0:N-1}\ Z^{0:N-1}) = P(S^N \mid S^{N-1}). \tag{1.6}$$

De plus, on fait l'hypothèse que la réponse du capteur à l'instant N est indépendant des états du système et des observations aux instants précédents. Le modèle capteur devient donc :

$$P(Z^N \mid S^{0:N}\ Z^{0:N-1}) = P(Z^N \mid S^N). \tag{1.7}$$

La décomposition 1.5 devient alors :

$$P(S^{0:N}\ Z^{0:N}) = P(S^{0:N-1}\ Z^{0:N-1})P(S^N \mid S^{N-1})P(Z^N \mid S^N). \tag{1.8}$$

En appliquant la même méthode au terme récursif, on obtient la décomposition finale :

$$\begin{aligned} &P(S^{0:N}Z^{0:N}) \\ &\quad = \left| \begin{array}{l} P(S^0)P(Z^0 \mid S^0) \\ \times \prod_{k=1}^{N} \left[\begin{array}{l} P(S^k|S^{k-1}) \\ \times P(Z^k|S^k) \end{array} \right] \end{array} \right. \end{aligned} \tag{1.9}$$

1.5.3 Formes paramétriques

Dans la décomposition 1.9, le terme $P(S^0)$ représente la connaissance dont dispose le programmeur sur l'état du système à l'instant $k = 0$. Si aucun a priori n'est connu, cette distribution de probabilités est choisie uniforme.
Les formes paramétriques des distributions de probabilités élémentaires restantes sont données soit par le modèle dynamique soit le modèle capteur.

La description est maintenant terminée, nous pouvons l'utiliser au moyen de questions.

1.5.4 Questions

Afin de réaliser la boucle décrite par la figure 1.5, deux questions sont posées à cette description à chaque instant k.
La phase de prédiction projette vers l'instant k l'état du système à partir des obser-

vations des instants 0 à $k-1$. La question à poser est donc :

$$P(S^k \mid z^{0:k-1}). \tag{1.10}$$

La phase d'estimation à l'instant k utilise les observations des instants 0 à k pour estimer l'état du système. La question à poser est donc :

$$P(S^k \mid z^{0:k}). \tag{1.11}$$

En pratique, la phase de prédiction à l'instant k s'appuie sur le résultat de l'estimation à l'instant $k-1$ et sur le modèle dynamique. La phase d'estimation à l'instant k confronte cette prédiction avec la nouvelle observation capteur, et ainsi de suite.

La figure 1.6 résume le programme bayésien que nous venons de détailler.

Programme
- Description
  - Spécification
    - Variables Pertinentes
      - $S^0, \ldots, S^N$: variables d'états du système ;
      - $Z^0, \ldots, Z^N$: variables d'observations.
    - Décomposition
      - $P(S^{0:N}\ Z^{0:N})$
      - $= P(S^0)P(Z^0 \mid S^0)$
      - $\times \prod_{i=1}^{N} P(S^i \mid S^{i-1})P(Z^i \mid S^i)$
    - Formes Paramétriques
      - $P(S^i \mid S^{i-1}), i = 1 \ldots N$: modèle dynamique ;
      - $P(Z^i \mid S^i, i = 0 \ldots N$: modèle capteur ;
      - $P(S^0)$: uniforme
  - Identification :
    - pas d'identification
- Question :
  - $P(S^k \mid z^0 \ldots z^{k-1})$ pour la phase de prédiction à l'instant k ;
  - $P(S^k \mid z^0 \ldots z^k)$ pour la phase d'estimation à l'instant k.

Fig. 1.6 : Filtre bayésien : le programme

1.6 Contrôle sensori-moteur au moyen du filtre bayésien [Koi05]

Dans cette section notre objectif est de présenter le canevas, proposé par [Koi05], pour contrôler un système sensori-moteur.

Cette approche propose d'associer à chaque centre d'intérêt évoluant dans l'espace des mouvements du robot une extension du filtre bayésien, décrit en section 1.5. À un instant t, chaque filtre doit estimer la position de son centre d'intérêt, en déduire un comportement approprié et finalement proposer les commandes motrices adéquates.

La position du centre d'intérêt est estimée grâce au modèle dynamique et au modèle capteur du filtre bayésien. Pour la sélection du comportement et des commandes motrices, [Koi05] propose d'ajouter au filtre deux autres modèles : un modèle de comportement et un modèle moteur. La décomposition du filtre devient :

$$P(S^{0:N}Z^{0:N}B^{0:N}M^{0:N}) = \left| \begin{array}{l} P(S^0Z^0B^0M^0) \\ \times \prod_{k=1}^{N} \left[\begin{array}{l} P(S^k|S^{k-1}) \\ \times P(Z^k|S^k) \\ \times P(B^k|S^kB^{k-1}) \\ \times P(M^k|S^kB^kM^{k-1}) \end{array} \right] \end{array} \right. \tag{1.12}$$

Nous retrouvons à l'intérieur du produit en première et seconde ligne, le modèle dynamique et le modèle capteur décrit dans la section précédente. En quatrième ligne, le modèle moteur décrit une collection d'actions motrices simples et indépendantes : suivre et fuir un objet, avancer droit devant, etc. Chacune de ces actions est un comportement basique. Le modèle de comportement, la troisième ligne du produit, quant à lui fournit un mécanisme pour la sélection des comportements.

Le comportement que le robot doit exécuter est le résultat de la fusion des propositions de chaque filtre.

Ce filtre sensori-moteur est appelé *filtre élémentaire*, non pas pour traduire une quelconque simplicité dans sa sémantique, mais pour rappeler que chacun de ces filtres est un élément du programme bayésien global.

Schéma d'utilisation

Le filtre bayésien élémentaire peut être schématisé par la figure 1.7. À chaque instant k la phase de *prédiction* extrapole l'état du système à l'instant $k-1$. Basé sur les nouvelles observations Z_i^k et des résultats de la prédiction, la phase *sélection de comportement* propose le comportement le plus adapté par rapport à son contexte. La proposition de chaque filtre est alors fusionnée (cf. étape « F ») pour en extraire le comportement b_t le plus approprié à être exécuté. La phase d'*estimation* met à jour la prédiction sachant le comportement retenu. La phase de *sélection des commandes motrices* propose l'action motrice la plus adaptée au vu de l'état du système à l'instant k dans le contexte du filtre et du comportement sélectionné. Finalement, à l'étape « F », les propositions de chaque filtre sont fusionnées pour décider les commandes motrices a exécuter.

1.6.1 Variables

Le filtre étendu utilise des variables de portée locale et globale. La variable locale n'existe que dans le contexte du filtre et par conséquent est indicée : Z_i. La variable globale quand à elle est commune à tous les filtres et est généralement utilisée pour la fusion.

Comme pour le filtre bayésien décrit dans la section précédente, le filtre étendu utilise

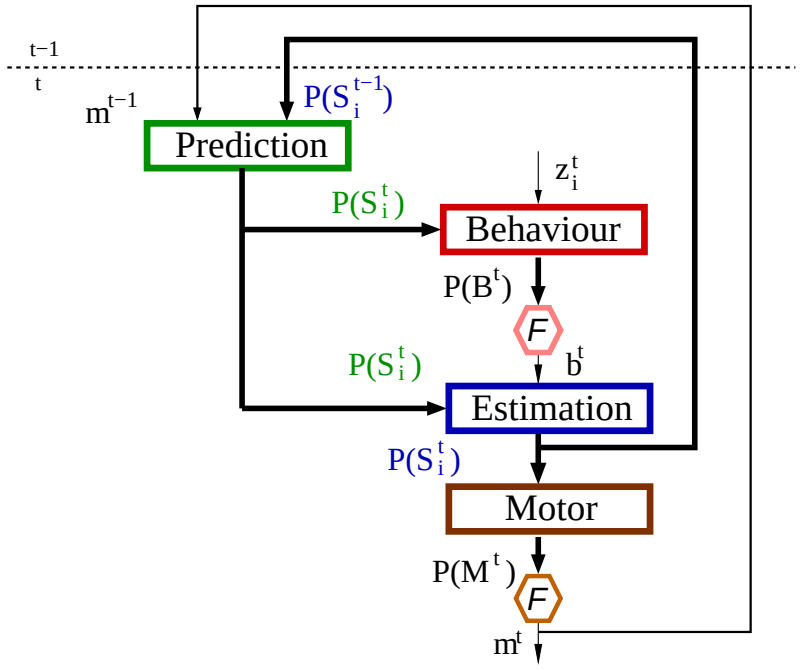


Fig. 1.7 : Boucle principale du filtre étendu [Koi05]

les variables locales d'observations Z_i et d'états S_i. Le modèle moteur introduit les variables motrices M. Elles sont globales car utilisées pour la fusion des commandes motrices.
Le modèle de comportement utilise un groupe de variables globales B. Chacune d'elles est associée à un groupe d'actions motrices.

Variables de cohérence

La fusion a été définie en section 1.4, comme une multiplication de distribution de probabilités. Ainsi, la fusion de comportement va consister à multiplier les propositions faites par chaque filtre sous forme d'une distribution de probabilité. Ce qui s'écrit sous la forme :

$$\begin{aligned} & P_G(B|\ldots) \\ & \quad = \; P_1(B|\ldots) \times P_2(B|\ldots) \times \ldots \times P_n(B|\ldots) \end{aligned} \tag{1.13}$$

D'après les règles énoncées en section 1.2.1, cette définition est incorrecte : la variable B apparaît plusieurs fois à gauche dans la décomposition. Ce problème peut être contourné en faisant appel à des variables intermédiaires, appelées également variables de cohérence. Á chaque variable B et M est associée respectivement une variable de cohérence locale β_i et λ_i. La fusion s'écrit alors sous la forme :

$$\begin{aligned} & P_G(B|\beta_i \ldots) \\ & \quad = \; P_1(\beta_1|B\ldots) \times P_2(\beta_2|B\ldots) \times \ldots \times P_n(\beta_n|B\ldots) \end{aligned} \tag{1.14}$$

1.6.2 Décomposition

En suivant le raisonnement de la section 1.5.2, la décomposition de la distribution conjointe pour un filtre peut s'écrire sous la forme :

$$\begin{aligned} & P(S^{0:t}Z^{0:t}M^{0:t}B^{0:t}\lambda^{0:t}\beta^{0:t}\pi_i) \\ & = \left| \begin{array}{l} \prod_{k=1}^{t} \left[\begin{array}{l} P(S_i^k|S_i^{k-1}\ M^{k-1}\pi_i) \\ \times P(Z_i^k|S_i^k\pi_i) \\ \times P(B^k|\pi_i) \times P(\beta_i^k|S_i^k B^k B^{k-1}\pi_i) \\ \times P(M^k|\pi_i) \times P(\lambda_i^k|S_i^k B^k M^k M^{k-1}\pi_i) \end{array} \right] \\ \times P(S_i^0 Z_i^0 B^0 M^0 \lambda_i^0 \beta_i^0|\pi_i). \end{array} \right. \end{aligned} \tag{1.15}$$

1.6.3 Formes paramétriques

Dans la décomposition 1.15, le terme $P(S_i^0 Z_i^0 B^0 M^0 \lambda_i^0 \beta_i^0|\pi_i)$ à l'extérieur du produit représente la connaissance dont dispose le programmeur sur l'état du système à l'instant $t = 0$. Si aucun a priori n'est connu, cette distribution de probabilités est choisie uniforme.
Les formes paramétriques des distributions de probabilités élémentaires à l'intérieur du produit sont données par le modèle dynamique, le modèle capteur, le modèle de comportement ou le modèle moteur.

Le modèle dynamique

$$P(S_i^k|S_i^{k-1}\ M^{k-1}\pi_i) \tag{1.16}$$

Il décrit l'évolution des variables d'état dans le temps. Les variables d'état à l'instant k dépendent de celles de l'instant S^{k-1} et des mouvements du robot au pas d'avant M^{k-1}.

Le modèle capteur

$$P(Z_i^k|S_i^k\pi_i) \tag{1.17}$$

Il définit la relation entre les observations obtenues des capteurs et les variables d'état.

Le modèle de comportement

$$P(B^k|\pi_i) \times P(\beta_i^k|B^k B^{k-1} S_i^k \pi_i) \tag{1.18}$$

Le comportement B^k à l'instant k dépend du comportement sélectionné au pas de temps précédent et des variables d'état. La dépendance avec B^{k-1} a pour objectif d'assurer une continuité (appelée persistance) dans l'exécution d'un comportement. La relation avec S^k garantit la réactivité aux changements de l'environnement.

Le modèle moteur

$$P(M^k|\pi_i) \times P(\lambda_i^k|S_i^k M^{k-1} B^k \pi_i) \tag{1.19}$$

La variable de comportement B^k est utilisée pour basculer entre les différents motifs moteurs : suivre et fuir un objet, rester immobile, avancer droit devant, etc. En considérant une variable de comportement B^t avec n valeurs possibles, nous obtenons le modèle moteur suivant :

$$\begin{aligned} &P(M^t|S_i^t B^t M^{t-1} \pi_i) \\ &\quad = \begin{cases} P(M^t|S_i^t M^{t-1}[B^t = b_1]\pi_i); \\ P(M^t|S_i^t M^{t-1}[B^t = b_2]\pi_i); \\ \dots \\ P(M^t|S_i^t M^{t-1}[B^t = b_{n_b}]\pi_i). \end{cases} \end{aligned} \tag{1.20}$$

Chaque terme de la décomposition ci-dessus correspond à un motif moteur :
- $P(M^t|S_i^t M^{t-1}[B^t = b_1]\pi_i)$ est le modèle moteur du comportement $b1$;
- $P(M^t|S_i^t M^{t-1}[B^t = b_2]\pi_i)$ est le modèle moteur du comportment $b2$, et ainsi de suite.

La description est maintenant terminée, nous pouvons l'utiliser au moyen de questions.

1.6.4 Questions

Afin de réaliser la boucle décrite par la figure 1.7, quatre questions sont posées à cette description à chaque instant k.

Prédiction

La phase de *prédiction* projette vers l'instant k l'état du système à partir des observations, des variables de comportement et des commandes motrices des instants 0 à $k-1$. La question à poser est donc :

$$P(S_i^k \mid z_i^{0:k-1}\ b^{0:k-1}\ m^{0:k-1}\ \lambda_i^{0:k-1}\ \beta_i^{0:k-1}). \tag{1.21}$$

Sélection du comportement

La phase de *sélection du comportement* décide le comportement le plus approprié basé sur les résultats de la prédiction et les nouvelles observations de l'instant k. Seuls les cas cohérents sont pris en compte : $\beta_i^k = 1$. Ainsi la question de sélection de comportement est donc :

$$P(B^k \mid z_i^{0:k}\ b^{0:k-1}\ m^{0:k-1}\ \lambda_i^{0:k-1}\ \beta_i^{0:k}). \tag{1.22}$$

Estimation La phase d'*estimation* à l'instant k utilise les observations et la valeur du comportement sélectionnée de l'instant k pour mettre à jour le résultat de la

prédiction et ainsi estimer l'état du système. La question à poser est donc :

$$P(S_i^k \mid z_i^{0:k}\ b^{0:k}\ m^{0:k-1}\ \lambda_i^{0:k-1}\ \beta_i^{0:k}). \tag{1.23}$$

Sélection des commandes motrices

La phase *sélection des commandes motrices* décide les commandes motrices les plus appropriées pour l'état estimé du système et du comportement choisi à l'instant k. Seuls les cas cohérents sont pris en compte : $\lambda_i^k = 1$. La question à poser est donc :

$$P(M^k \mid z_i^{0:k}\ b^{0:k}\ m^{0:k-1}\ \lambda_i^{0:k}\ \beta_i^{0:k}). \tag{1.24}$$

En pratique, la phase de prédiction à l'instant k s'appuie sur le résultat de l'estimation à l'instant $k-1$ et sur le modèle dynamique. La phase d'estimation à l'instant k confronte cette prédiction avec la nouvelle observation capteur, et la variable de comportement et ainsi de suite.

La figure 1.8 résume le programme bayésien que nous venons de détailler.

Programme

Description

Spécification

Variables Pertinentes

Variables locales :

S_i	:	variables d'états du système ;
Z_i	:	variables d'observations ;
β_i, λ_i	:	variables de cohérences.

Variables globales :

B	:	variables de comportements ;
M	:	variables motrices ;

Décomposition

$$P(S_i^{0:t} Z_i^{0:t} M^{0:t} B^{0:t} \lambda_i^{0:t} \beta_i^{0:t} \pi_i)$$
$$= P(S_i^0 Z_i^0 B^0 M^0 \lambda_i^0 \beta_i^0 | \pi_i)$$
$$\times \prod_{k=1}^{t} \left[\begin{array}{l} P(S_i^k | S_i^{k-1}\ M^{k-1} \pi_i) \\ \times P(Z_i^k | S_i^k \pi_i) \\ \times P(B^k | \pi_i) \times P(\beta_i^k | B^k B^{k-1} S_i^k \pi_i) \\ \times P(M^k | \pi_i) \times P(\lambda_i^k | S_i^k M^{k-1} B^k \pi_i) \end{array} \right]$$

Formes Paramétriques

$P(S_i^k \mid S_i^{k-1}\ M^{k-1} \pi_i), k = 1 \dots N$	:	modèle dynamique ;
$P(Z_i^k \mid S_i^k \pi_i), k = 0 \dots N$	:	modèle capteur ;
$P(B^k \mid \pi_i) \times P(\beta_i^k \mid B^k B^{k-1} S_i^k \pi_i), k = 0 \dots N$	:	modèle de comportement
$P(M^k \mid \pi_i) \times P(\lambda_i^k \mid S_i^k M^{k-1} B^k \pi_i), k = 0 \dots N$	:	modèle moteur ;
$P(S_i^0 Z_i^0 B^0 M^0 \lambda_i^0 \beta_i^0 \mid \pi_i)$	:	uniforme.

Identification :

pas d'identification

Question :

$P(S_i^k \mid z_i^{0:k-1}\ b^{0:k-1}\ m^{0:k-1}\ \lambda_i^{0:k-1}\ \beta_i^{0:k-1})$	Phase de prédiction ;
$P(B^k \mid z_i^{0:k}\ b^{0:k-1}\ m^{0:k-1}\ \lambda_i^{0:k-1}\ \beta_i^{0:k})$	Phase de sélection de comportement ;
$P(S_i^k \mid z_i^{0:k}\ b^{0:k}\ m^{0:k-1}\ \lambda_i^{0:k-1}\ \beta_i^{0:k})$	Phase d'estimation ;
$P(M^k \mid z_i^{0:k}\ b^{0:k}\ m^{0:k-1}\ \lambda_i^{0:k}\ \beta_i^{0:k})$	Phase de sélection des commandes motrices.

Fig. 1.8 : Filtre bayésien étendu : le programme

1.7 Librairie $ProBt^{©}$

Les programmes bayésiens présentés tout au long de ce mémoire sont implémentés en utilisant la librairie $ProBt^{©}$. Elle a été développée par l'équipe de recherche *eMotion* (laboratoire GRAVIR), et est commercialisée par la société ProBayes[1]. Elle est également disponible gratuitement pour des utilisations destinées à la recherche et à l'enseignement.

$ProBt^{©}$ est une librairie dynamique écrite en C++, portable sur Windows, Linux, Unix ou MacOS. Son API (Application Programming Interface) fournit l'accès aux classes d'objet probabiliste et à un moteur d'inférence bayésien. Ce dernier est pourvu de plusieurs algorithmes brevetés pour réduire la complexité et les temps de calcul. Ceci se fait bien sûr au détriment de l'exactitude.

[1] http ://www.probayes.com

Illustrons son utilisation avec un programme bayésien simple décrit par la figure 1.9. Il a pour but d'implémenter un modèle capteur qui met à jour une variable d'état S à partir d'une observation Z. La variable d'état doit être égale à la variable d'observation avec une certaine incertitude pour prendre en compte l'imprécision du capteur. La forme paramétrique du modèle est donc une gaussienne centrée sur S et dont l'écart-type traduit l'incertitude de la mesure.

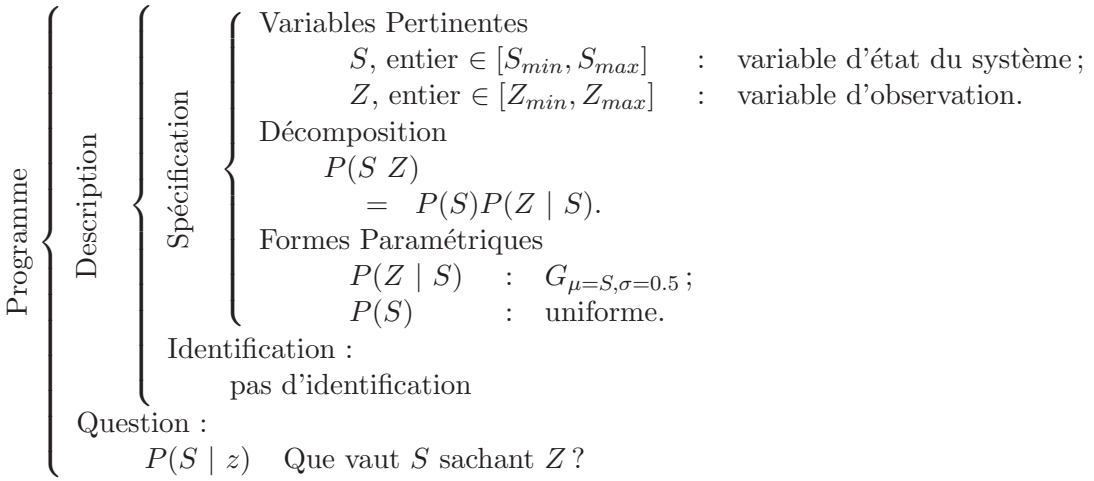


Fig. 1.9 : Exemple simple de programme bayésien

Déclaration des variables

La première étape est de déclarer les variables du modèle. S et Z sont des variables discrètes de type entier. Nous utilisons pour cela des *plSymbols* typés *plIntegerType*.

```
#include "pl.h"

//==== Déclarations des variables ======================
//
plIntegerType tZ(Z_MIN,Z_MAX) ;
plIntegerType tS(S_MIN,S_MAX) ;

plSymbol Z("Z", tZ) ;
plSymbol S("S", tS) ;
...
```

Formes paramétriques et distribution conjointe

L'étape suivante consiste à formuler les formes paramétriques, à construire la distribution conjointe et à établir la question.

Le terme $P(S)$ est choisi uniforme (*plUniform*) car la variable d'état peut prendre n'importe quelle valeur sans en favoriser une plus que les autres. Et comme nous l'avons spécifié précédemment, le terme $P(Z \mid S)$ est une gaussienne (*plCndBell-Shape*) qui permet de prendre en compte l'incertitude de la mesure.

La distribution conjointe est de type *plJointDistribution* qui prend pour paramètre de gauche la liste des variables du modèle, et comme paramètre de droite les termes de la décomposition.

```
...
//=== Formes paramétriques et décomposition =============
//=== P(S) :
plUniform P_S(S);

//=== P(Z | S) :
int sigma = 0.5;
plCndBellShape P_Z_S(Z, S, sigma);

//=== Distribution conjointe :
plJointDistribution pljd(Z ^ S, P_S * P_Z_S)
...
```

Utilisation

Nous disposons maintenant d'un programme bayésien complètement défini. La phase d'utilisation va consister à soumettre au modèle une nouvelle observation, invoquer le moteur d'inférence afin d'obtenir la distribution de probabilités sur S connaissant Z, et enfin sélectionner la valeur de S la plus appropriée. *ProBt*© propose plusieurs méthodes de sélection, des méthodes semi-aléatoires, avec ou sans limite de temps de calcul, etc. Dans notre exemple nous utilisons la méthode *best*, car elle extrait la valeur qui maximise la distribution de probabilité.

```
...
//=== Questions : ========================================
plCndKernel plKQ;
pljd.ask(plKQ, S, Z);

//=== Requête capteur d'une nouvelle observation : =========
plValue Zval = getCapteurObs();

//=== Utilisation : ======================================
//=== contruction de l'arbre des valeurs
plKernel plKI;
plKQ.instantiate(plKI, Zval);

//=== compilation
plKernel plKC;
plKI.compile(plKC);

//=== Sélection d'une valeur
plValue Sval;
plKC.best(Sval);

//=== Affichage du résultat ==============================
cout« "Pour Z="« Zval« " S="« Sval;
```

Cette librairie sera utilisée pour implémenter les programmes bayésiens présentés tout au long de ce document.

1.8 Synthèse

Nous avons introduit dans ce chapitre la programmation bayésienne comme un outil formel pour le développement de logiciels, capable de traiter les informations incomplètes et incertaines. Nous avons également présenter les techniques probabilistes, la fusion bayésienne et le filtre bayésien, sur lesquelles repose le canevas pour le contrôle d'un système sensori-moteur proposé par [Koi05]. Ce canevas sera utilisé pour implémenter le comportement spécifié dans l'introduction de ce document, sur le robot BIBA, que nous allons présenter dans le chapitre suivant.

Chapitre 2

Présentation de la plate-forme expérimentale

2.1 Le robot BIBA

Le robot BIBA, illustré sur la figure 2.1, est un robot mobile entièrement autonome. Il a été conçu par la société suisse *Bluebotics*[1] suivant le cahier des charges du projet européen BIBA. Il embarque une couche logicielle (cf. section 2.1.2) gérant les accès aux données des capteurs et aux commandes motrices. Nous détaillons maintenant ses aspects physiques.

2.1.1 Le matériel

Le robot BIBA est de taille moyenne (50 cm) et pèse environ 30 kg. Il est équipé d'une caméra montée sur un système pan-tilt et d'un télémètre laser situé sur l'avant du robot. Il est également équipé de capteurs de proximité infrarouge et à ultrasons. Les forces d'inertie appliquées au robot peuvent être mesurées grâce à un système vestibulaire. Saillantes au châssis, des plaques tactiles de sécurité ont été disposées tout autour du robot.

Le robot est pourvu de deux moteurs indépendants. Un sur la roue gauche, l'autre sur la roue droite. Une roue libre est placée sur l'avant du robot.

Le robot dispose d'un bloc avec un bus fond de panier « Compact PCI ». Sur ce bus, sont connectés une carte alimentation, deux cartes entrées-sorties et deux calculateurs : un PowerPC[2] « PowerCore CPCI-3750 » et un « PC Inova IPC-PM ». Des entrées-sorties, digitales et analogiques, RS422 et I2C, offrent également la possibilité d'étendre le système.

Le robot est équipé d'un mini réseau local. Le point d'accès est un HUB quatre ports intégrant un module sans fils configuré pour accéder au réseau de l'INRIA. Ceci permet de monitorer le robot à distance et de travailler plus confortablement sur un bureau en déporté tout en ayant accès aux capteurs et aux commandes motrices.

[1] www.bluebotics.com

[2] http ://fr.wikipedia.org/wiki/PowerPC

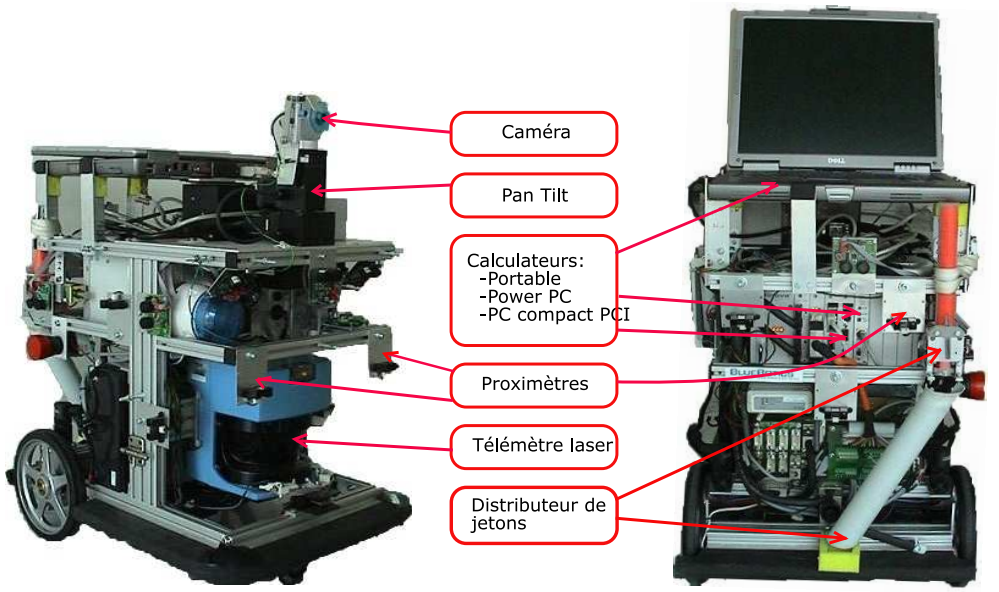


Fig. 2.1 : Le robot Biba développé par *Bluebotics*(www.bluebotics.com)

2.1.1.1 Les capteurs

Nous présentons, dans cette partie, un peu plus en détails les principaux capteurs utilisés dans notre expérimentation.

La caméra

La caméra est montée sur un système pan-tilt. Elle est orientable de $-160°$ à $+160°$ de gauche à droite et de $-46°$ à $+31°$ de bas en haut. Ses principales caractéristiques sont recensées dans le tableau 2.1.

Caractéristiques	Valeurs
Constructeurs	Orange Micro
Modèle	IBot
Type d'interface	IEEE1394 - Firewire 6 broches
Profondeur des couleurs	24 bits (16,7 millions de couleurs)
Type de capteur optique	1/4" CCD
Vitesse de la capture vidéo numérique	30 images/s
Résolution de la capture vidéo	640 x 480
Champ visuel	$\approx 60°$

Tab. 2.1 : Fiche technique de la caméra : ORANGE MICRO - IBOT

Elle est utilisée dans notre application comme principale source d'information sur l'environnement du robot.

Le télémètre laser

C'est un télémètre SICK LMS200, fixé à l'avant du robot. Il balaye de droite à gauche sur 180° avec une résolution angulaire de 0.5°. Il fournit ainsi 361 mesures par balayage. Chaque valeur est comprise entre 0 et 8192. Le télémètre peut ainsi mesurer des distances de 0 à $\tilde{8}0$ m avec une précision de 10 mm.

Dans notre application, ce capteur sera utilisé principalement pour la détection d'obstacles. Il est fiable, robuste et précis, presque parfait pour cette tâche sensible. Nous disons "presque" car son balayage est en 2D et du fait de son positionnement horizontal à 30 cm du sol, les obstacles qui ont des parties saillantes en dessous ou au dessus de cette hauteur peuvent poser problème : par exemple, les chaises qui ont un pied central, les tables, des objets suspendus etc.

2.1.1.2 Les ressources de calcul

Le robot est équipé de trois calculateurs hétérogènes, recensés sur la figure 2.2. Il a été livré d'origine avec le PowerPC. Les deux autres ont été ajoutés ultérieurement pour les besoins de cette expérience.

Le PowerPC, nommé *bibabot*, ne dispose pas de disque dur pour stocker le système d'exploitation. *bibabot* est donc configuré pour télécharger un fichier de démarrage automatiquement à la mise sous tension du robot. Il recherche ce fichier sur un serveur préalablement identifié, en utilisant le protocole TFTP[3]. *Bluebotics*, fournisseur du fichier de démarrage, a fait le choix d'utiliser *XO/2*[4] comme système d'exploitation temps réel.

Par nécessité de puissance de calcul, un deuxième calculateur CompactPCI « PC Inova IPC-PM », nommé *bibabot2*, a été ajouté. Contrairement au PowerPC, celui-ci est équipé d'une carte graphique, d'un disque dur, de trois ports USB, de deux ports série, de deux cartes ethernet et une entrée PS/2. Il est doté d'un microprocesseur à fréquence variable (600 Mhz à 1.6 Ghz) et de 760 Mo de mémoire RAM.

Enfin, le robot a été aménagé pour accueillir un PC portable, nommé *bibabotp*. Il a été choisi et équipé de façon à pouvoir prendre en charge la partie traitement vidéo. Pour cela notre choix s'est porté sur un « Dell Inspiron », équipé d'un processeur Pentium M750 à 2.0 Ghz, de 1 Go de mémoire RAM et d'une carte PCMCIA trois ports 1394 avec une alimentation externe.

Nous avons fait le choix d'utiliser *Linux* comme système d'exploitation. Ainsi *bibabot2* et *bibabotp* s'exécutent sur distribution Linux Debian Sarge[5].

2.1.1.3 Alimentation du robot

Le robot est alimenté par 2 batteries "Yuasa NPC17-12 12V 17AH". Il a été livré avec deux paires de batteries numérotées "set 1" et "set 2". Pour les préserver, il est recommandé de les utiliser par set. Lorsque le robot est en mouvement, l'autonomie est approximativement de deux heures. Il faut compter huit heures pour une recharge

[3]http ://fr.wikipedia.org/wiki/TFTP
[4]http ://xo2.org
[5]http ://www.debian.org/

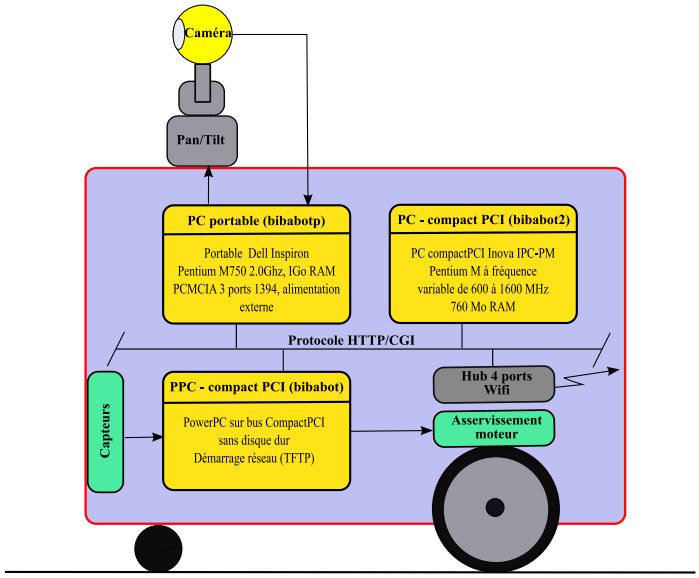


Fig. 2.2 : Les calculateurs embarqués

complète.
Le robot peut être également branché au secteur EDF via un transformateur 12V. Un interrupteur permet alors de basculer le mode d'alimentation. Ceci est très pratique pour changer de jeux de batteries sans avoir à éteindre le robot et repartir sur les phases d'initialisation. Ce mode est également utile pour préserver les batteries lors de travaux sur le robot à l'arrêt.

2.1.1.4 Alarme.

Le robot BIBA dispose d'une lampe flash ainsi que d'une alarme sonore qui se déclenche lors d'une anomalie du système : le robot est entré en contact avec un obstacle, le niveau des batteries est faible. Le mode alarme est géré par le contrôleur de bas niveau (section 2.1.2.1).

2.1.2 Le logiciel

Le robot a été livré avec une couche logicielle de bas niveau proposant une interface commune pour l'accès aux commandes motrices et aux données sensorielles via un contrôleur matériel. Cette couche s'exécute sur le PowerPC (*bibabot*).

Mais avant de détailler l'interface, listons les services fournis par le contrôleur de bas niveau.

2.1.2.1 Le contrôleur de bas niveau

Son rôle est de gérer les ressources matérielles du robot et d'exécuter des requêtes motrices ou sensorielles tout en garantissant sa sécurité et celle de son environnement.

Le contrôleur de bas niveau est composé de deux modules : l'un pour les déplacements du robot, l'autre pour la securité.

Position controller* et *Speed controller

Le contrôleur bas niveau propose deux méthodes pour le déplacement du robot :

- un contrôle en position. L'utilisateur donne la position cible (X, Y, théta) que le robot doit atteindre. Une fois la requête exécutée, le robot s'immobilise. L'origine est la position initiale du robot lors de l'initialisation du *position controller*. Il est important de noter que le robot perd sa référence dans le cas où il est déplacé manuellement : en débrayant les moteurs, en soulevant le robot, etc.
- un contrôle en vitesse. L'utilisateur donne la vitesse de la roue gauche et droite en Rad/s. Le *speed controller* maintient la commande jusqu'à la prochaine requête de l'utilisateur. Ainsi différent du déplacement en position, l'utilisateur doit explicitement envoyer une vitesse nulle comme requête pour immobiliser le robot.

Security controller

Ce module assure un minimum de sécurité pour le robot et son environnement. Il garantit que la vitesse limite n'est jamais dépassée. Il surveille la charge de la batterie, et met en route la lampe flash lorsque le niveau est trop faible (22 Volts). Si un problème est détecté, il stoppe instantanément le robot et le met en état d'alarme. L'alarme est visuelle et sonore.

Le *security controller* dispose de deux modes de sécurité face aux obstacles : le *BumperMode* et le*AvoidMode*. En *AvoidMode*, le robot évite les obstacles en utilisant les données du télémètre laser. Il conserve sa direction, mais s'arrête lorsque la distance à l'obstacle est inférieure au seuil minimum (50 cm). En *bumper mode*, l'évitement d'obstacle est désactivé. Le contact peut alors arriver. Quand cela se produit, le *security controller* stoppe le robot, coupe les amplificateurs moteurs et déclenche l'état d'alarme. Ce mode doit être choisi pour pouvoir utiliser le déplacement contrôlé en vitesse.
Le changement de mode est possible uniquement quand le *security controller* est stoppé. Pendant cette phase aucune commande ne peut être envoyée aux moteurs.

2.1.2.2 Interface HTTP/CGI

Le logiciel embarqué sur le PowerPC implémente une couche d'abstraction de bas niveau. Les données sensorielles, les commandes motrices et la configuration du robot sont accessibles via une interface HTTP/CGI[6]. La syntaxe d'une requête est :

http ://*RobotCtrl*/cgi/*nomModule.nomCGI* ?*Param1*+*Param2*+ . . . + *ParamN*

Les CGI sont regroupés par module. Il en existent deux principaux : *PPCtoPC* et *Preferences*. Ainsi *nomCGI* doit appartenir à *nomModule*. *RobotCtrl* est l'adresse IP ou le nom du PowerPC.

[6]Common Gateway Interface (http ://fr.wikipedia.org/wiki/CGI)

Nous pouvons noter qu'une requête est une adresse URL[7] tout à fait valide. Ce qui veut dire qu'elle peut être exécutée à partir d'un navigateur internet, comme illustré par la figure 2.3.

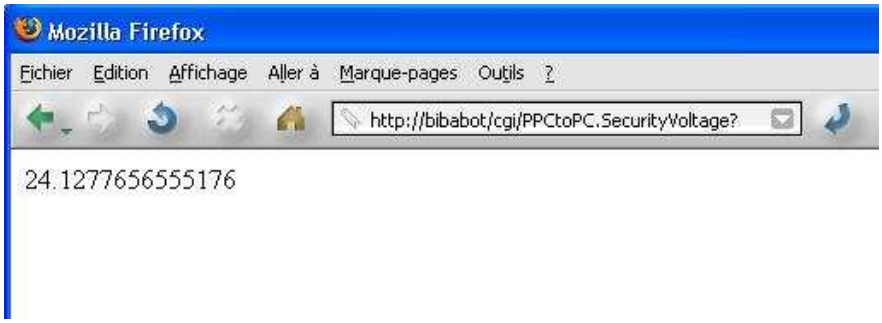


Fig. 2.3 : Utilisation de l'interface de bas niveau via un navigateur internet

Le module *PPCtoPC*

Les CGI donnant accès aux données des capteurs et des moteurs sont regroupés sous un seul module XO/2 : *PPCtoPC*.

La liste exhaustive des commandes est disponible dans [Blu04]. Cependant, les plus utilisées sont données dans le tableau 2.2. Par exemple, pour obtenir le niveau de la batterie, il faut envoyer au contrôleur la commande :

http ://bibabot/cgi/PPCtoPC.securityVoltage ?

Ce module ne supporte pas la caméra, ni le système pan-tilt et le capteur inertiel. Ces capteurs peuvent être accédés via la couche logicielle fournie par le système d'exploitation ou par le constructeur.

Le module *Preferences*

Un deuxième module, nommé *Preferences*, regroupe toutes les commandes pour la configuration du Robot. Les principales sont données dans le tableau 2.3.
Ce module permet entre autres de spécifier le nom du fichier de démarrage ainsi que l'adresse du serveur TFTP l'hébergeant. Une autre commande importante est : *SetDefaultHost*. Elle permet de spécifier le client par défaut, qui pourra envoyer des commandes motrices et des requêtes sensorielles sans avoir à s'authentifier. Depuis les autres machines ou pour changer la configuration, une authentification est nécessaire. Les commandes *SetUserName* et *SetUserPassword* définissent le *login*.

[7] uniform resource locator

Commandes	**Description**
bumpersGet	Retourne l'état des quatre contacts.
lsGetRawData	Renvoie le résultat d'un scan laser.
usGetAll	Retourne la liste des distances mesurées par les capteurs à ultrasons.
itGetAll	Renvoie la liste des distances mesurées par les capteurs infra-rouge.
speedSetSpeed	Définit la vitesse de rotation de la roue gauche et droite en rad/s.
positionGoto	Définit la position à atteindre, en coordonnées absolue (x, y , theta).
securityStart	Démarre le *security controller*.
securityStop	Stoppe le *security controller*. Aucune commande motrice n'est exécutée. Dans cet état il est possible de basculer du mode *bump* au mode *avoid*.
securitySetAvoidMode	Active l'évitement d'obstacles intégré au contrôleur de bas niveau. Mode à utiliser pour le contrôle en position.
securitySetBumpMode	Inhibe l'évitement d'obstacles. Les *bumpers* sont pris en compte. Mode à utiliser pour le contrôle en vitesse.
securityVoltage	Renvoie le niveau de la batterie en volts.
ioDoutXWRITE	Applique un état 0 ou 1 à la sortie numérique n°X.
ioDinXREAD	Lit et retourne l'état de l'entrée numérique n°X.
ioAinXREAD	Lit et retourne le niveau de l'entrée analogique n°X.
flashON	Active la lampe flash.
flashOFF	Inhibe la lampe flash.
i2cDoutWRITE	Permet de basculer ON/OFF une des huit sorties numériques.

Tab. 2.2 : Principales commandes du module *PPCtoPC*

Commandes	**Description**
Preferences.Show	Affiche la configuration courante.
Preferences.SetBootfileName	Nom du fichier de démarrage.
Preferences.SetServerAddress	Adresse IP du serveur TFTP hébergeant le fichier de démarrage.
Preferences.SetDefaultHost	Adresse IP du client par défaut. Pas d'authentification nécessaire.
Preferences.SetUserName Preferences.SetUserPassword	*Login* pour un accès administrateur ou utilisateur d'une autre machine que le *DefaultHost*.

Tab. 2.3 : Principales commandes du module *Preferences*

2.2 Extension des ressources

La plupart des extensions on été réalisées en collaboration avec l'équipe SED[8]. Ce groupe est composé majoritairement de membres permanents avec des compétences variées : électronique, développement logiciel, vision etc. L'équipe SED est une source d'information et une aide technique précieuse pour le développement des expériences à l'INRIA.

La liste des extensions effectuées pendant le stage sont recensées dans le tableau 2.4.

Accessoires	**commande**
Le pointeur laser	Sortie digitale 5.
Le poseur de jeton	Sortie digitale 4.
Contacteur recharge externe	Entrée digitale 3.
Bouton radio	Entrée digitale 5.

Tab. 2.4 : Accessoires ajoutés au robot

Le pointeur laser

Un pointeur laser a été fixé dans l'axe de visée de la caméra. De cette façon, lorsqu'il est activé, un point lumineux rouge apparaît sur l'objet au centre du champ de vision de la caméra. Il est commandé par la sortie digitale n°5 du robot.
Un obturateur voile le rayon laser quand il y a des risques d'éblouissement, comme illustré sur les photos 2.4.

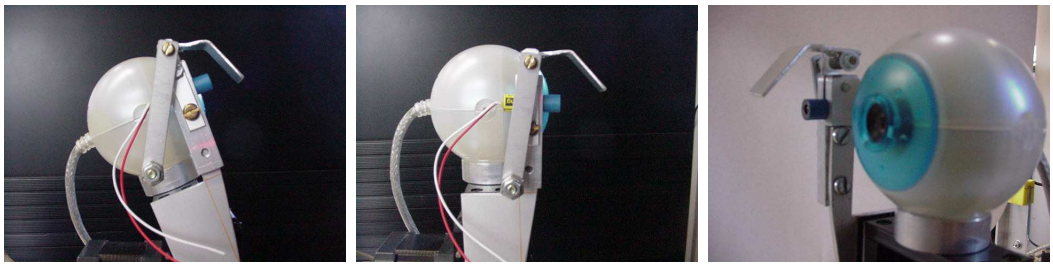


Fig. 2.4 : Robot : Le pointeur laser et sa protection anti-éblouissement

Le poseur de jetons

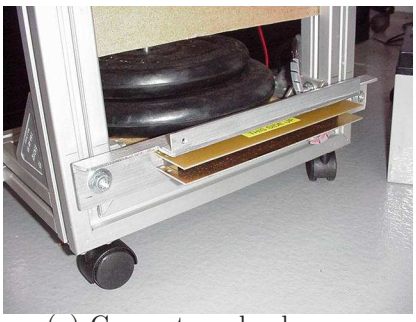
Le robot a été agrémenté d'un poseur de jetons dans le but de laisser des traces sur son parcours. Ce dispositif est composé d'un réservoir de jetons et d'un mécanisme de dépôt de jetons au centre de la trajectoire (photo 2.5). La commande du mécanisme est connectée à la sortie digitale $n°4$ du robot. Cet accessoire est prévu pour programmer un comportement de retour au point de départ.

[8]Supports expérimentaux et développement logiciel.

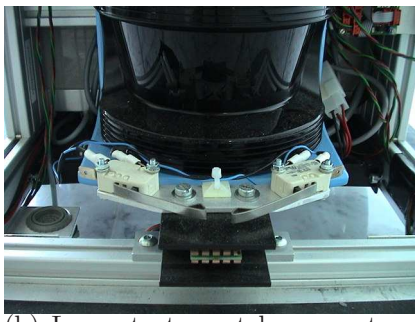
Fig. 2.5 : Bibabot et le poseur de jetons

Le contacteur de recharge

Ce dispositif informe le système si le robot est en charge. Il est connecté à l'entrée digital *n°3* du robot. Ce mécanisme est composé de deux boutons poussoirs positionnés de façon symétrique par rapport au connecteur de recharge (photos 2.6). Ceci permet d'être tolérant sur l'alignement du robot avec le chargeur.



(a) Connecteur du chargeur



(b) Le contacteur et le connecteur de recharge du robot

Fig. 2.6 : Bibabot et son contacteur de recharge

Le radio bouton

Le robot a été équipé d'un bouton poussoir radio. Le récepteur de ce dernier est connecté à l'entrée digitale *N°5* du robot.

2.3 Synthèse

Les mensurations du robot BIBA sont appropriés au type d'expérimentation que nous souhaitons réaliser. En effet, Il doit interagir avec des personnes qui jouent les rôles de prédateur, de maître et de proie, il ne pouvait donc pas être trop impressionnant par sa taille, ni trop petit. Pendant la fête de la science 2005 à l'INRIA, nous avons présenté le robot au public et réalisé des démonstrations de son comportement. Les gens l'ont trouvé plutôt attachant, surtout les plus petits.

Cependant le véritable atout du robot est sa panoplie de capteurs de technologies différentes : infrarouge, ultrason, laser, vision, inertiel, etc. Nous retenons surtout la précision du télémètre laser : 10 millimètres.

Chapitre 3

Comportement animal sur le robot BIBA

Nous allons, dans ce chapitre, décrire l'implémentation d'un comportement animal sur le robot BIBA. L'implémentation est réalisée en utilisant le canevas, « contrôle sensori-moteur au moyen du filtre bayésien » (section 1.6), proposé par [Koi05]. Nous commençons par spécifier le comportement désiré.

3.1 Principes du comportement du robot

Le comportement du robot est défini par analogie avec un comportement animal.

Le robot doit pouvoir se déplacer librement dans un environnement fermé tout en évitant les obstacles. Il interagit avec quatre centres d'intérêts : un prédateur, une proie, son maître et une route de fuite.

Face à un prédateur, il tente de se protéger. Près de son ennemi, il cherche à s'échapper, si une route de fuite est trouvée. Éloigné de celui-ci , il reste immobile pour ne pas être vu.

En présence du maître, le robot se sent en sécurité. Il ne fuit pas les éventuels prédateurs. Le robot suit son maître tant qu'il le perçoit. Si le robot détecte une proie il la prend en chasse, et lorsqu'elle est assez proche il la « capture ».

3.2 Implémentation sur le robot BIBA

Nous allons décrire dans cette section, l'utilisation des ressources du robot (ses capteurs et ses actionneurs) pour s'interfacer avec son environnement et la distribution des tâches (vision et calculs probabilistes) sur les calculateurs embarqués.

Les capteurs

Le prédateur, la proie et le maître sont associés à des couleurs et sont détectés par l'analyse du flux vidéo de la caméra : le prédateur est rouge, la proie verte et le maître bleu. Des humains joueront ces rôles en portant des maillots colorés comme

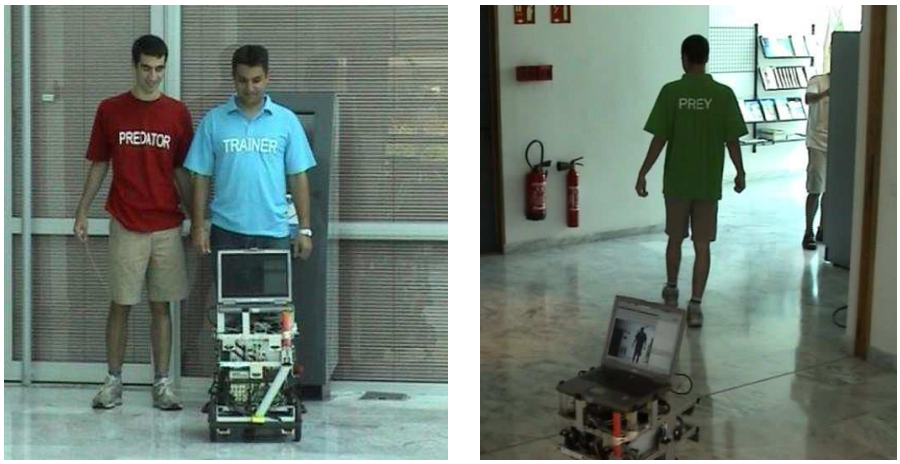


Fig. 3.1 : À gauche, le robot face au prédateur et au maître, pourchassant la proie à droite

illustré sur la figure 3.1 : à gauche, le robot face au prédateur et au maître, et pourchassant la proie à droite.

L'analyse du flux vidéo consiste à délimiter sur chaque image traitée des zones de ces couleurs. À partir de ces observations, la présence, la direction et la distance peuvent être estimées pour chaque acteur. La section 5.7 explicite l'algorithme de traitement vidéo.

La route de fuite est calculée à partir des données du télémètre laser. La direction retenue est celle pour laquelle la plus grande distance sans obstacles a été détectée. L'algorithme utilisé est une adaptation de [Man03] et [MAK05]. Pour éviter les routes de fuites trop étroites pour le robot, une fenêtre minimale de 15° balaye par pas de 0.5° les mesures de distances scannées par le télémètre Sick. La route de fuite est également soumise à une distance minimale paramétrable, en dessous de laquelle la fuite n'est pas considérée possible.

Les obstacles sont également détectés par le télémètre laser. Une couche sécurité d'évitement d'obstacles est détaillée au chapitre 4.

Les actionneurs

L'exécution d'un comportement revient à contrôler les vitesses de rotation des roues gauche et droite (variables M_g et M_d). Mais il est plus intuitif de contrôler le robot en vitesse de translation (variable M_{trans}) et vitesse de rotation (variable M_{rot}). La transformation est définie par les équations suivantes :

$$M_g = M_{trans} + M_{rot}$$
$$M_d = M_{trans} - M_{rot}$$

Certains des comportements spécifiés ci-dessus sont difficiles à implémenter sur un robot réel. Par exemple, pour des raisons de sécurité et par manque de ressources le robot ne peut pas capturer et consommer la proie comme un animal le ferait. La capture est donc simulée par un pointeur laser (détaillé en 2.2). La photo 3.2 illustre

Fig. 3.2 : Robot capturant sa proie plutôt souriante

la capture. Le point lumineux rouge sur le maillot indique que le robot consomme sa proie.

Lorsque le robot est en fuite, il perd le contact visuel avec le prédateur. Dans cette situation bien particulière, le modèle dynamique du filtre ne suffit pas à mémoriser la présence du prédateur pendant un temps suffisamment long pour terminer la fuite. Pour renforcer l'effet mémoire, une observation supplémentaire est utilisée par le biais d'un compte à rebours.

Les calculateurs

Les tâches les plus gourmandes en ressources de calcul sont le traitement vidéo et les calculs probabilistes. Le portable (*bibabotp*) est choisi pour s'occuper de toute la partie vision : contrôle de la caméra et du système pan-tilt, de l'acquisition et du traitement d'images.

Les calculs probabilistes sont exécutés par *bibabot2*, le calculateur « compactPCI » d'Inova. Il communique avec *bibabotp* pour récupérer les observations de la caméra, et avec *bibabot*, le « PowerPC », pour accéder aux données du télémètre laser. Cette distribution de tâches est illustrée par la figure 3.3.

3.3 Le programme bayésien

Le robot est contrôlé par quatre filtres sensori-moteurs élémentaires (décrits en section 1.6) : prédateur, proie, maître et route de fuite. La spécification des quatre

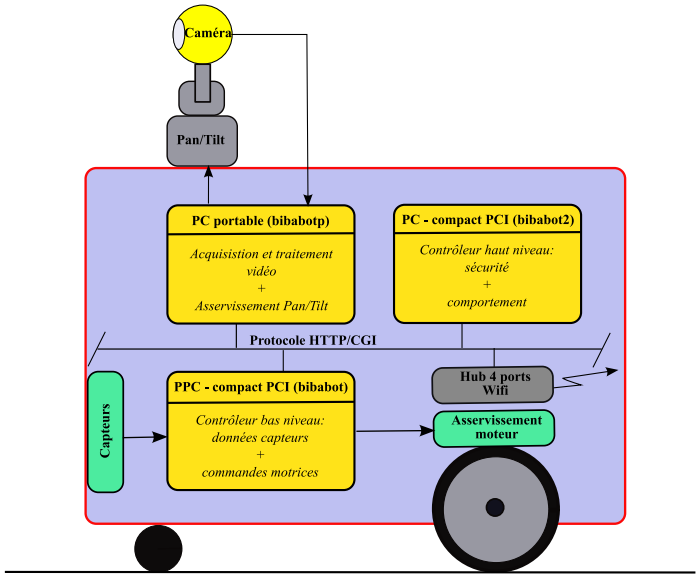


Fig. 3.3 : Distribution des tâches sur les calculateurs

filtres est, à quelques variantes près, similaire. Ce qui caractérise principalement chaque filtre est le choix des formes paramétriques et l'identification de leurs paramètres. Nous allons donc dans un premier temps décrire les aspects communs aux filtres élémentaires, puis nous détaillerons le filtre maître comme exemple, ensuite nous recenserons les spécificités des filtres proie, prédateur et route de fuite.

3.3.1 Le filtre élémentaire

3.3.1.1 Variables

Les variables globales

Les commandes motrices sont communes à tous les filtres. Elles sont composées d'une vitesse de rotation M_{rot} et d'une vitesse de translation M_{trans}.

M_{rot} est de type entier $[-4, +4]$. Pour les valeurs négatives le robot tourne à gauche, à droite pour les positives, et garde sa direction pour 0.

M_{trans} est de type entier $[0, 5]$. Le robot s'immobilise pour la valeur 0, et se déplace à la vitesse maximum autorisée pour 5.

La variable de comportement B^t est également commune à tous les filtres. Elle est de type entier et peut prendre une des valeurs *Escape*, *Motionless*, *Obey*, *Chase* et *Wander*.

Escape signifie que le prédateur est tout proche et que le robot doit chercher une route de fuite pour la suivre aussi rapidement que possible.

MotionLess, implique de rester immobile car le prédateur se trouve suffisamment éloigné pour espérer ne pas être vu.

Obey signifie qu'il faut adapter la vitesse pour suivre le maître à une distance relativement proche.

Chase implique de suivre la proie et d'adapter la vitesse pour la rattraper. Une fois très proche, la capture est simulée avec le pointeur laser.

Wander signifie d'avancer tout droit à vitesse moyenne, tout en évitant les obstacles.

Enfin, nous introduisons la variable globale $Zpan$, qui correspond à l'orientation de la caméra, donnée par le système pan-tilt. Elle permet de savoir si le centre d'intérêt est dans le champ de vision du robot. Cette information est importante pour définir correctement le modèle capteur. Comme illustré par la figure 3.4, le robot ne voit pas le maître. Cela ne veut pas dire qu'il n'est pas présent. Par contre il est certain que le prédateur est bien là. En résumé, le modèle capteur est sûr de ses observations uniquement pour les positions qui sont dans son champs visuel.

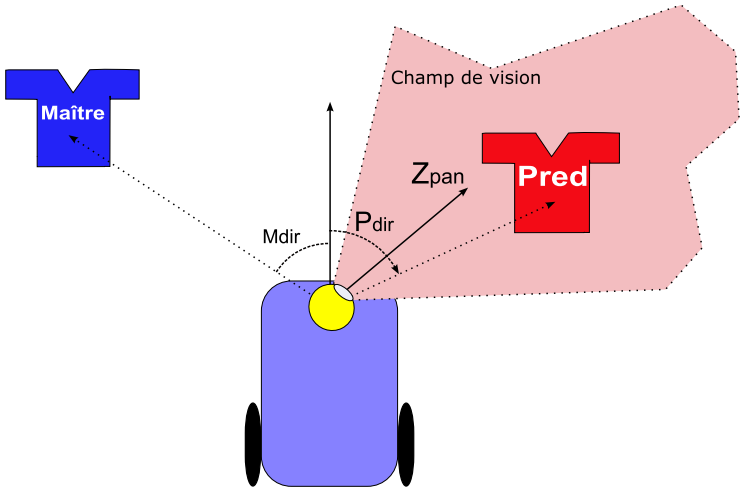


Fig. 3.4 : Z_{pan} donne l'orientation de la caméra

Les variables locales

Dans les variables locales à chaque filtre nous retrouvons les variables d'observations et d'états. À un instant k, un filtre doit estimer la présence, la direction et la distance du centre d'intérêt qui lui est associé : le prédateur pour le filtre prédateur, etc.

Présence est de type entier avec deux valeurs possibles $[0, 1]$. 1 indique que le centre d'intérêt est présent.

Distance est de type entier et peut prendre six valeurs $[0, 5]$: 0, le centre d'intérêt est très proche, 5 il est loin.

Direction est de type entier et neuf valeurs sont possibles $[-4, +4]$. Chaque valeur correspond à une zone autour du robot (voir figure 3.5). La valeur 0 indique que le centre d'intérêt se trouve droit devant le robot, pour les valeurs négatives il se trouve sur sa gauche, et sur sa droite pour les positives. Ainsi sur la figure, la direction de la proie est -2, la direction du prédateur est 1 et celle du maître est 4.

Pour finir, à chaque variable globale fusionnée doit être associée une variable de cohérence (explication donnée en section 1.6.1) : λ^k_{rot} et λ^k_{trans} sont associées respecti-

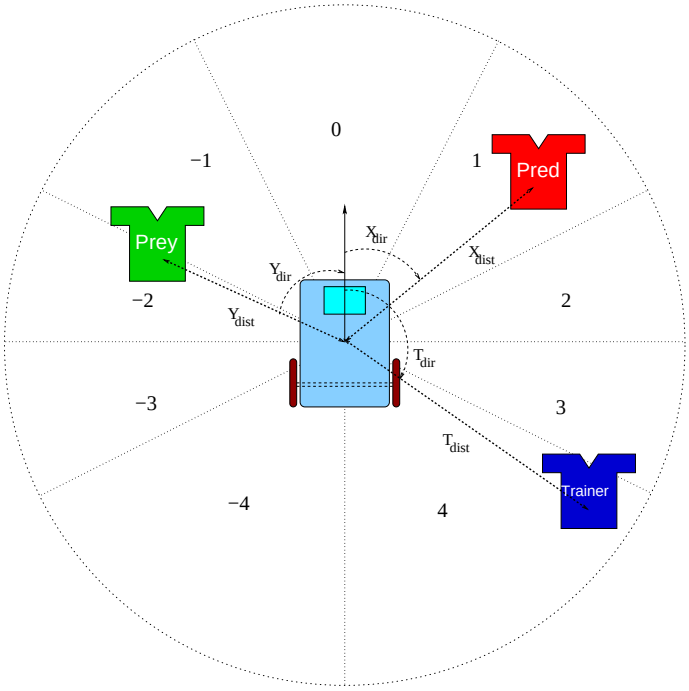


Fig. 3.5 : Convention pour estimer la direction des objets

vement aux commandes motrices M^k_{rot} et M^k_{trans}, et β^k à la variable de comportement B^k.

3.3.1.2 Décomposition

La décomposition de la distribution conjointe du filtre élémentaire (équation 3.1) a pour modèle celle du filtre sensori-moteur (section 1.6.2). Elle est composée de plusieurs termes : le terme d' « initialisation » du système à l'instant $k = 0$, décrit à l'extérieur du produit, et à l'intérieur du produit le modèle dynamique (vert), le modèle capteur (bleu), le modèle de comportement (rouge) et le modèle moteur (marron).

$$
\begin{aligned}
&P(Si^{0:t} Zi^{0:t} Z_{pan} M^{0:t}_{rot} M^{0:t}_{trans} B^{0:t} \lambda i^{0:t}_{rot} \lambda i^{0:t}_{trans} \beta i^{0:t} \pi_i) \\
&= \left| \prod_{k=1}^{t} \left[\begin{array}{l}
P(Si^k_{pres} | Si^{k-1}_{pres} \pi_i) \times P(Si^k_{dist} | Si^{k-1}_{dist} B^{k-1} \pi_i) \\
\times P(Si^k_{dir} | Si^{k-1}_{dir} B^{k-1} M^{k-1}_{rot} \pi_i) \\
\times P(Z^k_{pan} | \pi_i) \times P(Zi^k_{pres} | Si^k_{pres} Si^k_{dir} Z^k_{pan} \pi_i) \\
\times P(Zi^k_{dist} | Si^k_{dist} Zi^k_{pres} Si^k_{dir} Z^k_{pan} \pi_i) \\
\times P(Zi^k_{dir} | Si^k_{dir} Zi^k_{pres} Z^k_{pan} \pi_i) \\
\times P(B^k | \pi_i) \times P(\beta i^k | B^k B^{k-1} Si^k_{pres} Si^k_{dist} \pi_i) \\
\times P(M^k_{rot} | \pi_i) \times P(\lambda i^k_{rot} | M^k_{rot} B^k Si^k_{pres} Si^k_{dir} \pi_i) \\
\times P(M^k_{trans} | \pi_i) \times P(\lambda i^k_{trans} | M^k_{trans} B^k Si^k_{pres} Si^k_{dist} \pi_i)
\end{array} \right] \right. \\
&\left. \times P(Si^0 Zi^0 Z^0_{pan} B^0 M^0_{rot} M^0_{trans} \lambda i^0_{rot} \lambda i^0_{trans} \beta i^0 | \pi_i). \right.
\end{aligned}
\tag{3.1}
$$

3.3.1.3 Formes paramétriques et identification des paramètres

Le modèle dynamique ne prend pas en compte la dynamique propre aux centres d'intérêts. Seul le comportement et les mouvements du robot sont utilisés.

Le modèle de comportement favorise le comportement dominant du filtre si son centre d'intérêt est présent, et l'interdit s'il est absent. Par exemple le filtre maître recommande fortement les comportement *Obey* si le maître est détecté. Toutefois le comportement dominant de certains filtres doit être confirmé par un autre filtre. C'est le cas du comportement *Escape* du filtre prédateur, qui est viable seulement si une route de fuite a été trouvée par son filtre.

Le modèle moteur implémente les actions motrices pour le(s) comportement(s) dominant(s) du filtre. De cette façon, le filtre maître implémente le comportement « suivre » le maître. *Wander* n'est un comportement dominant pour aucun filtre. Il est cependant le comportement par défaut du robot. Pour cela, chaque filtre implémente l'action moteur pour exécuter ce comportement.

3.3.2 Le filtre *maître*

3.3.2.1 Variables

Les variables d'observations et d'états correspondent à la présence, la direction et la distance du maître. On définit les groupes de variables suivants :

$$Sm^k = \left\{Sm^k_{pres}, Sm^k_{prox}, Sm^k_{dir}\right\}$$

$$Zm^k = \left\{Zm^k_{pres}, Zm^k_{prox}, Zm^k_{dir}\right\}$$

Les variables de cohérence locale au filtre maître sont : λm^k_{rot} et λm^k_{trans} sont associées respectivement aux commandes motrices M^k_{rot} et M^k_{trans}, et βm^k à la variable de comportement B^k.

3.3.2.2 Décomposition

La décomposition de la distribution conjointe pour ce filtre s'écrit :

$$\begin{aligned}
&P(Sm^{0:t} Zm^{0:t} Z_{pan} M^{0:t}_{rot} M^{0:t}_{trans} B^{0:t} \lambda m^{0:t}_{rot} \lambda m^{0:t}_{trans} \beta m^{0:t} \pi_m) \\
&= \left| \prod_{k=1}^{t} \left[\begin{array}{l}
P(Sm^k_{pres} | Sm^{k-1}_{pres} \pi_m) \times P(Sm^k_{dist} | Sm^{k-1}_{dist} B^{k-1} \pi_m) \\
\times P(Sm^k_{dir} | Sm^{k-1}_{dir} B^{k-1} M^{k-1}_{rot} \pi_m) \\
\times P(Z^k_{pan} | \pi_m) \times P(Zm^k_{pres} | Sm^k_{pres} Sm^k_{dir} Z^k_{pan} \pi_m) \\
\times P(Zm^k_{dist} | Sm^k_{dist} Zm^k_{pres} Sm^k_{dir} Z^k_{pan} \pi_m) \\
\times P(Zm^k_{dir} | Sm^k_{dir} Zm^k_{pres} Z^k_{pan} \pi_m) \\
\times P(B^k | \pi_m) \times P(\beta m^k | B^k B^{k-1} Sm^k_{pres} Sm^k_{dist} \pi_m) \\
\times P(M^k_{rot} | \pi_m) \times P(\lambda m^k_{rot} | M^k_{rot} B^k Sm^k_{pres} Sm^k_{dir} \pi_m) \\
\times P(M^k_{trans} | \pi_m) \times P(\lambda m^k_{trans} | M^k_{trans} B^k Sm^k_{pres} Sm^k_{dist} \pi_m)
\end{array} \right] \right. \\
&\left. \times P(Sm^0 Zm^0 Z^0_{pan} B^0 M^0_{rot} M^0_{trans} \lambda m^0_{rot} \lambda m^0_{trans} \beta m^0 | \pi_m). \right.
\end{aligned} \tag{3.2}$$

3.3.2.3 Formes paramétriques et identification des paramètres

Modèle dynamique

Le modèle dynamique est composé de trois termes : un pour chaque variable d'état :

$$= \dots \prod_{k=1}^{t} \left[\begin{array}{l} P(Sm^k_{pres}|Sm^{k-1}_{pres}\pi_m) \times P(Sm^k_{dist}|Sm^{k-1}_{dist}B^{k-1}\pi_m) \\ \times P(Sm^k_{dir}|Sm^{k-1}_{dir}B^{k-1}M^{k-1}_{rot}\pi_m) \\ \dots \end{array} \right] \quad (3.3)$$

Pour la *présence*, le modèle dynamique est une dilution dans le temps. Le tableau 3.1 montre que le modèle dynamique tend à conserver l'état d'avant : les probabilités les plus fortes sont pour $Sm^k_{pres} = Sm^{k-1}_{pres}$. Par conséquent, si le maître n'était pas présent au pas de temps d'avant il ne l'est *probablement* toujours pas, mais nous n'en sommes pas sûrs. Cette incertitude se traduit par des valeurs de probabilité assez proches : 0.6 et 0.4.

Sm^k_{pres} \ Sm^{k-1}_{pres}	0	1
0	0.6	0.3
1	0.4	0.7

Tab. 3.1 : Modèle dynamique du maître – *présence* : $P(Sm^k_{pres}|Sm^{k-1}_{pres}\pi_m)$

Le modèle dynamique pour la *distance* est donné par le tableau 3.2. Il traduit l'idée suivante : si le robot était en train de suivre le maître ($B^{k-1} = obey$), alors il y a de fortes chances que la distance ait diminué, car le robot adapte sa vitesse pour rattraper son maître. Pour les autres comportements, la dynamique est incertaine, il est toutefois privilégié de conserver la même distance. D'où, comme forme paramétrique, une courbe « bell-shaped » centrée sur Sm^{k-1}_{dist} avec un écart-type important pour traduire l'incertitude.

B^{t-1} Sm^{t-1}_{dist} \ Sm_{dist}	O						E,M,C,W
	0	1	2	3	4	5	0..5
0	x	x	1e-1	1e-2	1e-3	1e-4	$G_{Sm^{t-1}_{dist},\sigma}$
1	1e-1	0.3	x	1e-1	1e-2	1e-3	
2	1e-2	1e-1	0.3	x	1e-1	1e-2	
3	1e-3	1e-2	1e-1	0.3	x	1e-1	
4	1e-4	1e-3	1e-2	1e-1	0.3	x	
5	1e-5	1e-4	1e-3	1e-2	1e-1	0.3	

Tab. 3.2 : Modèle dynamique du maître – *distance* : $P(Sm^k_{dist}|Sm^{k-1}_{dist}B^{k-1}\pi_m)$, $\sigma = 0.5$. E, M, C, O, W sont les valeurs de B, et signifie respectivement : *Escape*, *Motionless*, *Chase*, *Obey* et *Wander*.

Le filtre dynamique pour la *direction*, doit corriger la direction du maître par rapport au mouvement du robot au pas de temps $k-1$. Si le robot ne suit pas le

maître ($B^{k-1} \neq obey$) la forme paramétrique de ce terme est une gaussienne pour augmenter à chaque pas de temps l'incertitude sur la direction du maître.

Dans le cas où le robot suit le maître ($B^{k-1} = obey$), nous introduisons une constante K qui correspond au δ_{dir} effectué par le robot en un pas de temps à une vitesse de rotation $M_{rot} = 1$. En multipliant k par la vitesse de rotation M_{rot}^{k-1}, on obtient le delta déplacement en rotation Δ_{rot} parcouru par le robot.

La forme paramétrique de ce terme et une gaussienne centrée sur $Sm_{dir}^{k-1} + K * Mm_{rot}^{k-1}$ et avec un écart-type de 0.5 pour prendre en compte l'incertitude de la prédiction. Le filtre dynamique pour la direction est résumé par le tableau 3.3.

Sm_{dir} \ $Sm_{dir}^{k-1} B^{k-1}$	1	2-4
	-4..4	-4..4
-4..4	$G_{\mu=Sm_{dir}^{t-1}+K*M_{rot}^{k-1},\sigma=0.5}$	$G_{\mu=Sm_{dir}^{t-1},\sigma=0.5}$

Tab. 3.3 : Modèle dynamique du maître – *direction* : $P(Sm_{dir}^{k}|Sm_{dir}^{k-1}B^{k-1}M_{rot}^{k-1}\pi_m)$

Modèle capteur

Le modèle capteur est composé d'un terme par variable d'observations : présence, distance, direction du maître et la position du système pan-tilt Z_{pan} :

$$= \dots \prod_{k=1}^{t} \left[\begin{array}{l} \dots \\ P(Z_{pan}^{k}|\pi_m) \times P(Zm_{pres}^{k}|Sm_{pres}^{k} Sm_{dir}^{k} Z_{pan}^{k} \pi_m) \\ \times P(Zm_{dist}^{k}|Sm_{dist}^{k} Zm_{pres}^{k} Sm_{dir}^{k} Z_{pan}^{k} \pi_m) \\ \times P(Zm_{dir}^{k}|Sm_{dir}^{k} Zm_{pres}^{k} Z_{pan}^{k} \pi_m) \\ \dots \end{array} \right] \tag{3.4}$$

Le système pan-tilt est contrôlé par le module de vision et ne dépend pas des variables d'états des filtres sensori-moteurs. Nous n'avons aucun a priori sur des valeurs plus probables que d'autres pour Z_{pan}. Le terme $P(Z_{pan}|\pi_m)$ est donc choisi uniforme.

Les observations de présence, distance et direction du maître n'ont un sens que si le maître est dans le champ de vision du robot, défini par $|Sm_{dir} - Z_{pan}| \leq 2$.

Le tableau 3.4 définit le modèle capteur pour la *présence*. Nous remarquons que si le sujet n'est pas vu ($|Sm_{dir} - Z_{pan}| > 2$) alors aucun avis n'est donné : $P([zm_{pres} = 0]) = 0.5$ et $P([zm_{pres} = 1]) = 0.5$.

Zm_{pres} \ Sm_{pres} Sm_{dir} Z_{pan}	$\|Sm_{dir} - Z_{pan}\| \leq 2$		$\|Sm_{dir} - Z_{pan}\| > 2$
	0	1	0...1
0	0.7	0.1	0.5
1	0.3	0.9	0.5

Tab. 3.4 : Modèle capteur du maître – *présence* : $P(Zm_{pres}^{k}|Sm_{pres}^{k} Sm_{dir}^{k} Z_{pan}^{k} \pi_m)$

La variable d'état *distance* est mise à jour uniquement si le maître est présent ($Zm_{pres}^{k} = 1$) et dans le champ de vision du robot ($|Sm_{dir} - Z_{pan}|le2$). Dans les

autres cas, la forme paramétrique du terme $P(Zm^k_{dist}|Sm^k_{dist}Sm_Zm^k_{pres}dir^kZ^k_{pan}\pi_m)$ est choisie uniforme (voir tableau 3.5).

Sm_{dist} Sm_{dir} Z_{pan} Z_{pres} / Zm_{dist}	$\|Sm_{dir} - Z_{pan}\| \leq 2$		$\|Sm_{dir} - Z_{pan}\| > 2$	
	0	1	0	1
0..5	Uni	$G_{\mu=Sm_{dist},\sigma=0.25}$	Uni	Uni

Tab. 3.5 : Modèle capteur du maître – *distance* : $P(Zm^k_{dist}|Sm^k_{dist}Sm_Zm^k_{pres}dir^kZ^k_{pan}\pi_m)$.

L'observation de direction est pertinente si le maître est présent et dans la zone visible ($|Sm_{dir} - Z_{pan}| \leq 2$), autrement aucun avis n'est donné (uniforme).

Zm_{pres} Sm_{dir} Z_{pan} / Zm_{dir}	$\|Sm_{dir} - Z_{pan}\| \leq 2$		$\|Sm_{dir} - Z_{pan}\| > 2$	
	0	1	0	1
-4..4	Uni	$G_{\mu=Sm_{dir},\sigma=0.25}$	Uni	Uni

Tab. 3.6 : Modèle capteur du maître – *direction* : $P(Zm^k_{dir}|Sm^k_{dir}Zm^k_{pres}Z^k_{pan}\pi_m)$.

Modèle de comportement

Le modèle de comportement est un modèle inverse, nécessaire pour la fusion de la variable B^k. Cependant, le modèle direct est plus intuitif à écrire. C'est pourquoi le tableau 3.7 décrit l'implémentation du modèle de comportement sous sa forme directe.

$$= \ldots \prod_{k=1}^{t} \left[\begin{array}{l} \ldots \\ P(B^k|\pi_m) \times P(\beta m^k|B^kB^{k-1}Sm^k_{pres}Sm^k_{dist}\pi_m) \\ \ldots \end{array} \right] \tag{3.5}$$

Dans les deux premières colonnes de la table, nous remarquons que si le robot s'échappait ou était immobile et que le maître est maintenant présent, le comportement *Obey* est fortement recommandé. Ceci implémente le sentiment de sécurité que doit simuler le robot lorsqu'il voit son maître. Si ce dernier est absent, *Obey* est interdit et les autres comportements sont équiprobablement recommandés.

Modèle moteur

Le modèle moteur est composé de deux sous-modèles inverses : un pour la vitesse de rotation M^k_{rot} et un deuxième pour la vitesse de translation M^t_{trans}.

$$= \ldots \prod_{k=1}^{t} \left[\begin{array}{l} \ldots \\ P(M^k_{rot}|\pi_m) \times P(\lambda m^k_{rot}|M^k_{rot}B^kSm^k_{pres}Sm^k_{dir}\pi_m) \\ \times P(M^k_{trans}|\pi_m) \times P(\lambda m^k_{trans}|M^k_{trans}B^kSm^k_{pres}Sm^k_{dist}\pi_m) \\ \ldots \end{array} \right] \tag{3.6}$$

B^{k-1}	E,M		C			O			W	
Sm^k_{pres}	0	1	0	1		0	1		0	1
Sm^k_{dir} / B^k	*	*	*	0..3	4..5	*	0..3	4..5	*	*
E	+	-	+	-	-	+	-	+	+	-
M	+	-	+	-	-	+	-	+	+	-
C	+	-	+	+	++	+	+	+	+	-
O	×	++	×	++	++	×	++	++	×	++
W	+	×	+	×	×	+	×	×	++	×

Tab. 3.7 : Modèle direct de comportement du maître : $P(B^k|B^{k-1}Sm^k_{pres}Sm^k_{dist}\pi_m)$. E, M, C, O, W sont les valeurs de B, et signifie respectivement : *Escape*, *Motionless*, *Chase*, *Obey* et *Wander*. '++' signifie que le comportement est fortement recommandé, '+' recommandé, '-' pas recommandé et '×' interdit.

Raisonner sur un modèle direct est plus facile. C'est donc sous cette forme que les tableaux 3.8 et 3.9 illustrent respectivement les modèles pour M_{rot} et M_{trans}. Notons que seuls deux comportements sont pertinents pour le filtre maître :

- *Obey* qui est le comportement dominant. Il consiste à s'orienter vers le maître ($G_{\mu=S_{dir},\sigma=0.5}$) et adapter sa vitesse ($G_{\mu=Sm_{dist},\sigma=0.5}$) pour se rapprocher du maître ;
- et, *Wander*, le comportement par défaut du robot qui signifit avancer tout droit ($G_{\mu=0,\sigma=8}$) à vitesse moyenne ($G_{\mu=\frac{VTMAX}{2},\sigma=5}$) tout en évitant les obstacles.

Pour tous les autres, le filtre maître ne donne pas d'avis (Uni)

$B\ S_{dir}$ / M_{rot}	*Obey*	*Wander*	*
-4..4	$G_{\mu=S_{dir},\sigma=0.5}$	$G_{\mu=0,\sigma=8}$	Uni

Tab. 3.8 : Modèle direct moteur du maître – *vitesse de rotation* : $P(M^k_{rot}|B^kSm^k_{pres}Sm^k_{dir}\pi_m)$.

$B\ Sm_{dist}$ / M_{trans}	*Obey*	*Wander*	*
0..5	$G_{\mu=Sm_{dist},\sigma=0.5}$	$G_{\mu=\frac{VTMAX}{2},\sigma=5}$	Uni

Tab. 3.9 : Modèle direct moteur du maître – *vitesse de translation* : $P(M^k_{trans}|B^kSm^k_{pres}Sm^k_{dist}\pi_m)$.

3.3.3 Le filtre *proie*

Le filtre proie est très similaire au filtre maître. Si la proie est détectée et que le comportement choisi est *Chase*, alors le robot la suit dans le but de la capturer. La capture est simulée par l'activation du pointeur laser.

3.3.3.1 Variables

Le filtre proie utilise une nouvelle variable motrice pour actionner le pointeur laser. Nous la nommons Mp_{las}. Elle est de type entier avec deux valeurs possibles $[0, 1]$. Pour $mp_{las} = 1$ le pointeur laser est actionné.

Différente des autres variables motrices, elle est relative uniquement au filtre proie. Mp_{las} est donc locale et n'est pas fusionnée.

3.3.3.2 Décomposition

À la distribution conjointe du filtre élémentaire, le filtre proie ajoute au modèle moteur un nouveau terme pour contrôler le pointeur laser :

$$\begin{aligned} & P(Sp^{0:t} Zp^{0:t} Z_{pan} M^{0:t}_{rot} M^{0:t}_{trans} Mp^{0:t}_{las} B^{0:t} \lambda p^{0:t}_{rot} \lambda p^{0:t}_{trans} \beta p^{0:t} \pi_p) \\ & = \left| \begin{array}{l} \prod_{k=1}^{t} \left[\begin{array}{l} \dots \\ \times P(M^k_{rot}|\pi_p) \times P(\lambda p^k_{rot}|M^k_{rot} B^k Sp^k_{pres} Sp^k_{dir} \pi_p) \\ \times P(M^k_{trans}|\pi_p) \times P(\lambda p^k_{trans}|M^k_{trans} B^k Sp^k_{pres} Sp^k_{dist} \pi_p) \\ \times P(Mp^k_{las}|B^k Sp^k_{pres} Sp^k_{dir} Sp^k_{dist} \pi_p) \end{array} \right] \\ \times P(Sp^0 Zp^0 Z^0_{pan} B^0 M^0_{rot} M^0_{trans} Mp^0_{las} \lambda p^0_{rot} \lambda p^0_{trans} \beta p^0 | \pi_p). \end{array} \right. \end{aligned} \tag{3.7}$$

3.3.3.3 Formes paramétriques et identification des paramètres

La forme paramétrique du nouveau terme est une table de probabilités qui implémente l'idée suivant : le pointeur laser est activé si le robot est en train de chasser ($B^k = Obey$) et si la proie est présente ($Sp_{pres} = 1$), devant lui ($Sp_{dir} = 0$) et à une distance très courte ($Sp_{dist} = 0$) :

$$P([mp^k_{las} = 1]|[b^k = Obey][sp^k_{pres} = 1][sp^k_{dir} = 0][sp^k_{dist} = 0]\pi_p) = 1$$

Pour tous les autres cas :

$$P(Mp^k_{las}|B^k Sp^k_{pres} Sp^k_{dir} Sp^k_{dist} \pi_p) = 0$$

3.3.4 Le filtre *prédateur*

Le filtre prédateur présente deux comportements dominants : fuir (*Escape*) et rester immobile (*Motionless*). Lorsque le robot est en fuite, il perd le contact visuel avec le prédateur, car il essaie de s'échapper dans la direction opposée. Dans cette situation bien particulière, le modèle dynamique ne suffit pas à mémoriser la présence du prédateur pendant un temps suffisamment long pour continuer la fuite. Pour renforcer l'effet mémoire, une observation supplémentaire est utilisée par le biais d'un compte à rebours.

3.3.4.1 Variables

Une nouvelle variable, nommé Zr_{timer}, est ajoutée aux variables d'observations de ce filtre, pour faire persister la présence du prédateur lorsqu'il n'est plus dans le champ de vision. Elle est de type entier avec deux valeurs possibles $[0, 1]$. Tant que $Zr_{timer} = 1$ le robot mémorise que le prédateur est présent.

3.3.4.2 Décomposition

À la distribution conjointe du filtre élémentaire, le filtre prédateur modifie le modèle capteur en introduisant la variable Zr_{timer} dans le terme *présence* :

$$\begin{aligned} & P(Sr^{0:t} Zr^{0:t} Z_{pan} M_{rot}^{0:t} M_{trans}^{0:t} B^{0:t} \lambda r_{rot}^{0:t} \lambda r_{trans}^{0:t} \beta r^{0:t} \pi_r) \\ &= \left| \begin{array}{l} \prod_{k=1}^{t} \left[\begin{array}{l} \dots \\ \times P(Z_{pan}^k | \pi_r) \\ \times P(Zr_{pres}^k Zr_{timer}^k | Sr_{pres}^k Sr_{dir}^k Z_{pan}^k \pi_r) \\ \times P(Zr_{dist}^k | Sr_{dist}^k Zr_{pres}^k Sr_{dir}^k Z_{pan}^k \pi_r) \\ \times P(Zr_{dir}^k | Sr_{dir}^k Zr_{pres}^k Z_{pan}^k \pi_r) \\ \times \dots \end{array} \right] \\ \times P(Sr^0 Zr^0 Z_{pan}^0 B^0 M_{rot}^0 M_{trans}^0 \lambda r_{rot}^0 \lambda r_{trans}^0 \beta r^0 | \pi_r). \end{array} \right. \end{aligned} \tag{3.8}$$

3.3.4.3 Formes paramétriques et identification des paramètres

Modèle capteur

La forme paramétrique du nouveau terme est une table de probabilités dont l'implémentation est illustrée par le tableau 3.10. Afin de remplir la table, la première question qui se pose est : est-ce que le prédateur pourrait être dans le champ de vision du robot ($|Sr_{dir}^k - Z_{pan}k| \leq 2$) ou pas ($|Sr_{dir}^k - Z_{pan}k| > 2$) ?

Si le prédateur pouvait être visible, mais qu'il n'est pas présent alors il est fort probable que le capteur ne le voit pas (première colonne). Par contre s'il est présent (deuxième colonne), il est peu probable qu'aucune observation ne confirme sa présence.

Z_{timer} prend toute son importance quand le prédateur n'est pas visible. La dernière colonne montre l'effet mémoire apporté par Z_{timer} : le prédateur peut être présent sans que le robot ne le voit.

Sr_{dir}^k $Z_{pan}k$	$\lvert Sr_{dir}^k - Z_{pan}k\rvert \leq 2$		$\lvert Sr_{dir}^k - Z_{pan}k\rvert > 2$	
Zr_{pres}^k Zr_{timer}^k \ Sr_{pres}^k	0	1	0	1
0 0	0.35	0.02	0.49	0.001
0 1	0.35	0.3	0.01	0.333
1 0	0.15	0.34	0.49	0.333
1 1	0.15	0.34	0.01	0.333

Tab. 3.10 : Modèle capteur du prédateur – *Présence* : $P(Zr_{pres}^k Zr_{timer}^k | Sr_{pres}^k Sr_{dir}^k Z_{pan}^k \pi_r)$.

Modèle de comportement

Le filtre prédateur n'a pas un comportement dominant, comme pour les autres filtres : fuir, rester immobile et suivre le maître sont des comportements possibles lorsque le robot détecte la présence du prédateur. Fuir est une bonne solution si une route de fuite existe, et suivre le maître est judicieux seulement si le maître est présent. Ainsi, les propositions du filtre prédateur doivent être confirmées par d'autres filtres.

Le modèle de comportement est programmé selon le tableau 3.11. Il implémente les idées suivantes :

- Si le prédateur n'est pas présent, peu importe le comportement qu'il suivait, *Escape* et *Motionless* sont interdits et *Obey*, *Chase* et *Wander* sont équiprobables.
- Si le prédateur est présent, quelle que soit sa distance, et que le robot s'échappait, il est fortement recommandé qu'il continue de fuir pour se mettre a l'abri. *Obey* est également possible, mais doit être confirmé par le filtre du maître.
- Lorsque le prédateur est présent, le choix entre rester immobile ou s'échapper dépend de la distance à laquelle il se trouve. Loin du prédateur il préférera rester immobile pour espérer ne pas être vu. Dans tous les cas, suivre le maître reste possible si le filtre maître le suggère également.

B^{k-1}	E		M			C			O,W		
$Sr_{pres}k$	0	1	0	1		0	1		0	1	
$B^k \quad Sr_{dist}k$	*	*	*	0..3	4..5	*	0..3	4..5	*	0..3	4..5
E	×	++	×	++	+	×	++	+	×	++	+
M	×	-	×	-	++	×	-	++	×	-	++
C	+	-	+	-	-	+	-	+	+	-	-
0	+	+	+	+	+	+	+	++	+	+	++
W	+	×	+	×	×	+	×	×	+	×	×

Tab. 3.11 : Modèle de comportement du prédateur : $P(B^k|B^{k-1}Sr^k_{pres}Sr^k_{dist}\pi_r)$.E, M, C, O, W sont les valeurs de B, et signifient respectivement : *Escape*, *Motionless*, *Chase*, *Obey* et *Wander*. '++' signifie que le comportement est fortement recommandé, '+' recommandé, '-' pas recommandé et '×' interdit.

3.3.5 Le filtre *Route de fuite*

Le filtre route de fuite est le plus simple des filtres. Son unique rôle est de détecter une route de fuite et recommander ou pas la fuite. Le modèle de comportement se résume au tableau 3.12.

3.4 Conclusion : algorithme d'utilisation des filtres

Nous disposons maintenant de quatre filtres sensori-moteurs complètement définis. La phase d'utilisation va consister à mettre en oeuvre les filtres par le biais des

B^k \ Se^k_{pres}	0	1
E	×	+
M	○	○
C	○	○
O	○	○
W	○	○

Tab. 3.12 : Modèle de comportement du filtre route de fuite : $P(B^k|B^{k-1}Se^k_{pres}\pi_e)$. E, M, C, O, W sont les valeurs de B, et signifient respectivement : *Escape*, *Motionless*, *Chase*, *Obey* et *Wander*. '+' signifie recommandé, '○' pas d'avis et '×' interdit.

actions suivantes :

1. La question de prédiction est posée à chaque filtre élémentaire ;
2. Acquisition et prétraitement des données sensorielles, et mise à jour des variables d'observations Z ;
3. La question de comportement est posée à chaque filtre élémentaire en utilisant les observations définies à l'étape précédente.
4. Fusion des réponses obtenues, et tirage de la valeur pour la variable de comportement ;
5. La question d'estimation est posée à chaque filtre élémentaire ;
6. La question moteur est évaluée par chaque filtre élémentaire ;
7. Les réponses à la question moteur locale à chaque filtre sont alors fusionnées et de la distribution de probabilités globale résultante est sélectionnée une valeur pour chaque variable motrice ;
8. Enfin les variables des termes récursifs de la distribution conjointe sont mises à jour pour être utilisées aux pas de temps suivant.

Au terme de cette séquence de questions, nous avons des commandes motrices que le robot doit exécuter. Pour cela, les ordres moteurs sont envoyés au module d'évitement d'obstacles, qui doit certifier qu'ils peuvent être exécutés en toute sécurité. Les commandes résultantes sont alors envoyées au contrôleur de bas niveau.

Le programme bayésien d'évitement d'obstacles est décrit au chapitre suivant. Les résultats du contrôleur de comportements seront présentés au chapitre 6.

Chapitre 4

Évitement d'obstacles réactif

Le robot BIBA doit pouvoir évoluer dans un environnement non contrôlé. Pour sa sécurité et celle des personnes partageant son espace de mouvement, il doit être incapable d'exécuter des commandes dangereuses. C'est un prérequis élémentaire pour un robot mobile. Il doit éviter les obstacles statiques (murs, portes, plantes etc), ainsi que les obstacles dynamiques non agressifs (personnes, autres robots).

Cette section présente l'implémentation du module d'évitement d'obstacles. Son rôle est de transmettre au système d'asservissement de bas-niveau, autant que possible, les commandes choisies par le contrôleur de comportements, tout en garantissant la sécurité du robot.

4.1 Spécification

Rappelons que le robot ne possède pas de carte de l'environnement. L'évitement d'obstacles est donc purement réactif aux données sensorielles à un instant t.

Nous utilisons l'approche proscriptive proposée par [KPBM03]. Elle propose de transformer le problème d'évitement d'obstacles en fusion bayésienne de sous-modèles proscriptifs. Chaque sous-modèle construit une distribution de probabilités sur tous les cas des commandes motrices et variables d'observations. Les cas dangereux ont une valeur de probabilité proche de « 0 » et une valeur équiprobable proche de « 1 » pour les autres.

Le résultat de la fusion des sous-modèles est une distribution de probabilités intégrant les cas interdits et la commande choisie par le contrôleur de haut niveau. La commande motrice la plus probable de cette distribution est alors envoyée au système d'asservissement moteur.

Les commandes motrices sont composés d'une vitesse de translation Vt et d'une vitesse de rotation Vr. Nous notons Vtd et Vrd, les commandes désirées par le contrôleur de comportement.

4.1.1 Pré-traitement des données sensorielles

Cette méthode propose de découper le champ de "vision" du robot en n zones (cf figure 4.1). Le choix de n dépend fortement du nombre de données sensorielles et de la contrainte de temps de calcul.

À chaque secteur est associé une et une seule mesure de distance. Dans le cas où une zone dispose de plusieurs données sensorielles, la distance la plus courte sera conservée. Notons $D_k, k = 1 \dots n$, les variables probabilistes correspondant à ces mesures.

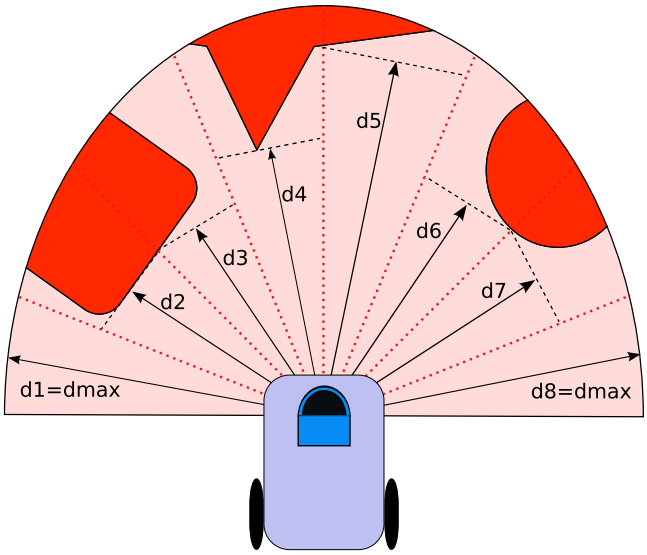


Fig. 4.1 : Évitement d'obstacles : variables d'observations

4.1.2 Le sous-modèle proscriptif

Dans un secteur angulaire i, on définit une distribution de probabilités sur (V_t, V_r) connaissant la distance D_i mesurée dans ce secteur.

$$P_i(V_t\ V_r \mid D_i) = P_i(D)P_i(V_t \mid D_i)P_i(V_r \mid D_i) \tag{4.1}$$

$P_i(D)$ est une loi uniforme, car les obstacles peuvent être rencontrés à n'importe quelle distance. De plus, on ne souhaite pas privilégier certaines distances par rapport à d'autres.
$P_i(V_t \mid D_i) \times P_i(V_r \mid D_i)$ exprime les commandes qui sont sûres au vu de la mesure de distance D_i. Ces deux termes s'appuient sur une sigmoïde en deux dimensions (éq. 4.2), pour avoir une distribution continue. Dans l'équation de la sigmoïde 2D, $F()$ est propre à chaque terme. Ainsi, on aura respectivement pour $P_i(V_t \mid D_i)$ et $P_i(V_r \mid D_i)$, $f_{trans}(V_t, D_i)$ et $f_{rot}(V_r, D_i)$. Ces deux fonctions résument le comportement du robot face à un obstacle.

$$SIG2D() = 1 - \frac{1}{1 + e^{-0.5\alpha(F()-d)}} \tag{4.2}$$

α a une influence forte sur la pente de la sigmoïde au point d'inflexion. Il est important de choisir une pente assez forte pour deux raisons : d'abord, cela permet d'exprimer le caractère impératif de la contrainte de sécurité. De cette façon, le respect des contraintes de sécurité est bien considéré comme une contrainte dure. Ensuite, une pente forte au point d'inflexion impose des plateaux très plats, ce qui permet de considérer de manière équiprobable toutes les commandes sans danger.

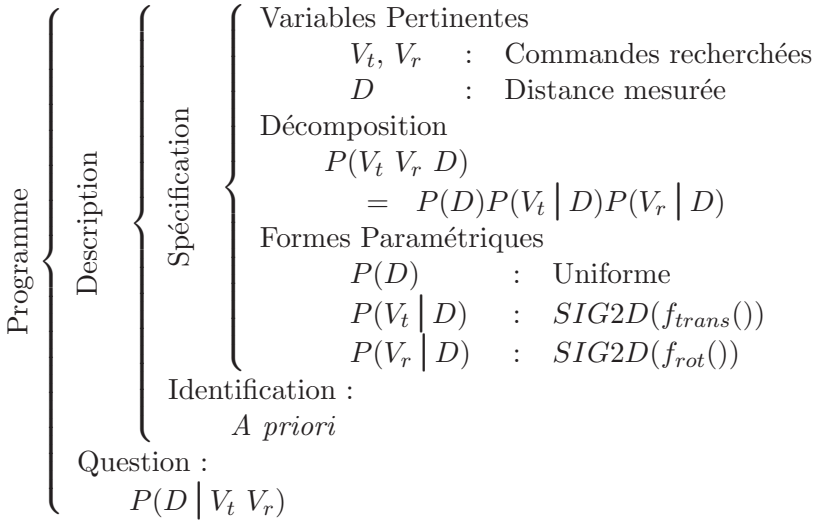


Fig. 4.2 : Évitement d'obstacles : sous-modèle bayésien proscriptif

4.1.3 Le sous-modèle "suivi de consigne"

La commande qui sera générée par le module d'évitement d'obstacles doit d'une part être sûre, et d'autre part respecter autant que possible les commandes désirées par le pilote de haut-niveau. Notons ces commandes V_{td} et V_{rd}. La distribution conjointe est :

$$P(V_t\ V_r \mid V_{td}\ V_{rd}) = P(V_{td}\ V_{rd})P(V_t \mid V_{td})P(V_r \mid V_{rd}) \tag{4.3}$$

où $P(V_{td}\ V_{rd})$ est une loi uniforme, car n'importe quelle commande peut être choisie de façon équiprobable par le pilote de haut-niveau.
$P(V_t \mid V_{td})$ et $P(V_r \mid V_{rd})$ sont des distributions courbes "bell-shaped" centrées respectivement sur V_{td} et V_{rd}.

$$P(V_t \mid V_{td}) = G_{(\mu=V_{td},\sigma_t)}$$
$$P(V_r \mid V_{rd}) = G_{(\mu=V_{rd},\sigma_r)}$$

Les écarts-types σ peuvent être vus comme des niveaux de contrainte. Le suivi de consigne étant une contrainte molle par opposition aux contraintes de sécurité, σ doit être important.

4.1.4 Fusion des sous-modèles

Pour fusionner l'ensemble des deux sous-modèles proscriptif et suivi de consigne, on utilise le programme bayésien illustré par la figure 4.3. Finalement, la commande appliquée est celle qui maximise la distribution de la fusion car elle répond au mieux à la commande désirée et aux contraintes de sécurité.

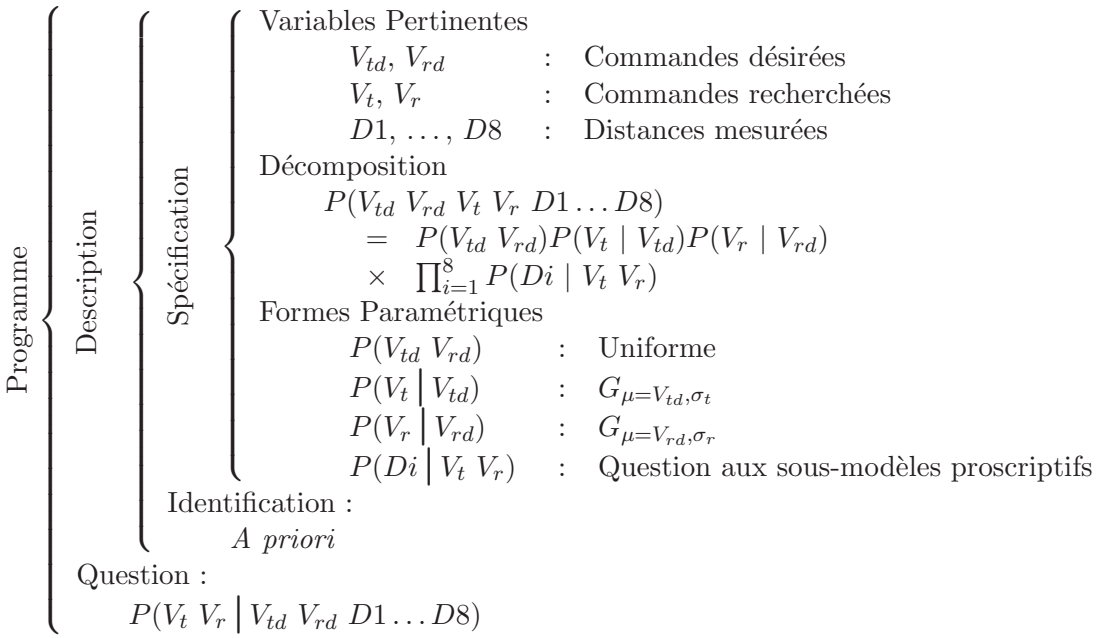


Fig. 4.3 : Évitement d'obstacles : fusion des sous-modèles

4.2 Identification des paramètres

Pour achever la description du modèle d'évitement d'obstacles, nous devons maintenant fixer la valeur des paramètres laissés libres dans les sous-modèles proscriptif et suivi de consigne. Nous fixons ces paramètres a priori.

4.2.1 Le sous-modèle proscriptif

Les paramètres restant à identifier dans le sous-modèle proscriptif sont les fonctions $f_{trans}()$ et $f_{rot}()$ respectivement les formes paramétriques des termes $P(V_t \mid D)$ et $P(V_r \mid D)$ du programme 4.2. Ces deux fonctions résument le comportement d'évitement d'obstacles suivant : plus l'obstacle est éloigné, plus on essaie de suivre la commande désirée, et inversement, plus l'obstacle est proche, plus la sécurité sera privilégiée.

Mais à partir de quelle distance l'obstacle est supposé trop proche ? Et inversement quand est-ce qu'il peut être considéré suffisamment éloigné pour suivre les commandes désirées ? D'où la nécessité de définir des seuils de contraintes.

Les seuils de contraintes : D_{min} et D_{max}

Chaque sous-modèle définit deux distances seuils (cf. figure 4.4) : D_{min} et D_{max}. Au dessus de D_{max}, l'obstacle est suffisamment éloigné pour permettre au robot de suivre la commande désirée. Le niveau de contrainte est nul. En dessous de D_{min}, l'obstacle est dans une zone interdite. Le robot cherche impérativement à s'en écarter. Il est peu probable qu'il se trouve dans cette situation car il se sera au préalable dévié de l'obstacle lorsque celui-ci était dans la zone à risque. De D_{min} à D_{max}, les contraintes de sécurité sont relâchées graduellement. Les courbes $f_{trans}()$ et $f_{rot}()$ définissent cette progression.

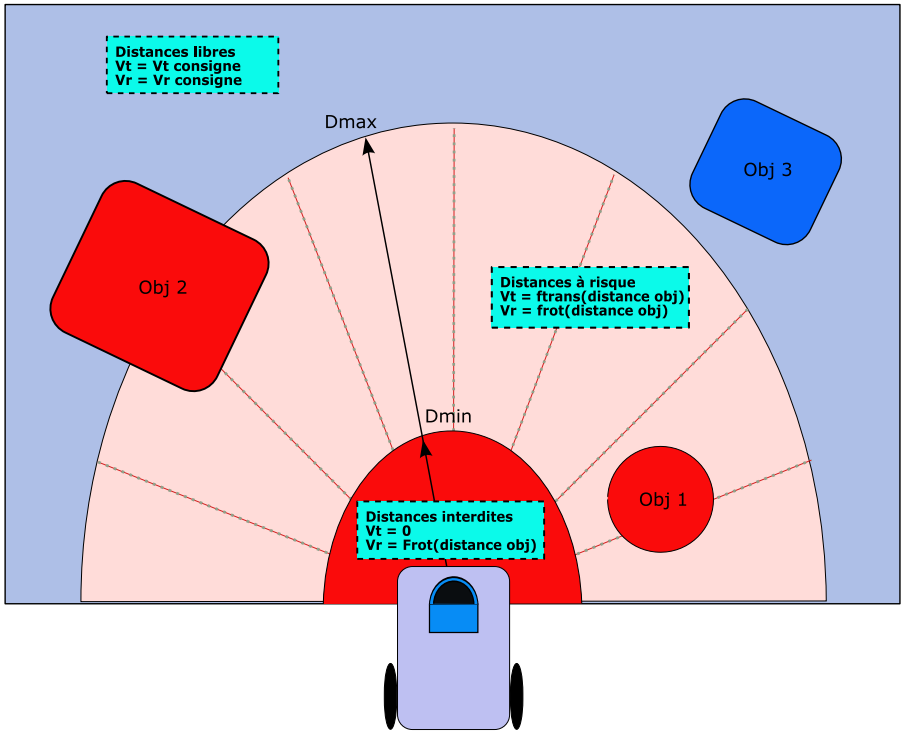


Fig. 4.4 : Évitement d'obstacles : les seuils de contraintes

Les seuils de contraintes dépendent principalement de deux facteurs : les caractéristiques du robot (son encombrement, sa vitesse de déplacement, son inertie, etc.) et l'environnement dans lequel il évolue (fort ou faiblement encombré, obstacles statiques et/ou dynamiques, vitesse des obstacles dynamiques, etc.).

Contraintes sur la vitesse de rotation : $P(V_r \mid D)$

$f_{rot}()$ doit implémenter l'idée suivante : plus l'obstacle est proche plus le robot cherche à s'en éloigner en tournant au maximum de ses possibilités. Si l'obstacle est à droite, le robot tournera à gauche et si l'obstacle est à gauche, il tournera à droite.

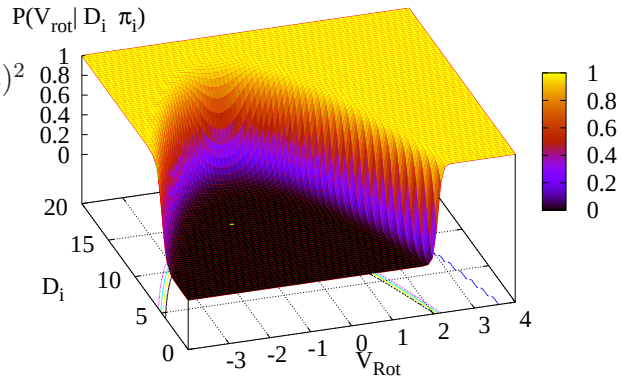
La courbe pour $P(V_r \mid D)$ de la zone 2 ainsi que l'équation $f_{rot}()$ sont donnés par le table 4.1. La zone 2 se trouve sur le côté gauche du robot(cf figure 4.1). Nous notons que pour des distances à l'obstacle inférieures au seuil D_{min} (50cm) ($D_i <= 5$)

les seules vitesses de rotation autorisées sont 2, 3 et 4. Le robot va donc tourner à droite. Pour des distances supérieures au seuil D_{max} (1,50m) aucune contrainte n'est appliquée à la consigne.

$$f_{rot}(V_r) = D_{max} - \frac{3}{2}\frac{D_{max}-D_{min}}{D_{max}}(V_r - \mu)^2$$

avec :

$\mu = -2$
$D_{min} = 5(50cm)$
$D_{max} = 15(1,50m)$



Tab. 4.1 : Évitement d'obstacles : f(Vrot, Distance)

La fonction $f_{rot}()$ est identique pour tous les sous-modèles proscriptifs. Cependant, toute zone, en présence d'obstacles, doit interdire au robot de s'orienter dans sa direction. Ainsi, la « fenêtre » des vitesses de rotation interdites se déplace progressivement de la gauche, pour les zones à gauche du robot, vers la droite, pour les zones à droite. Par conséquent, chaque zone doit connaître sa position relative au robot. Cette information est résumée dans le paramètre μ.

Contraintes sur la vitesse de translation : $P(V_t \mid D)$

$f_{trans}()$ doit implémenter l'idée suivante : quand un obstacle est très proche du robot, par mesure de sécurité, sa vitesse doit être impérativement contrainte à zéro. Inversement, quand l'obstacle est loin ou inexistant, le robot peut suivre la commande fournie par le pilote de haut-niveau.

La courbe pour $P(V_t \mid D)$ ainsi que l'équation $f_{trans}()$ sont données par le table 4.2. Nous notons que plus l'obstacle est proche, plus la vitesse de translation est réduite et pour des distances à l'obstacle inférieures à 50 cm ($D_i <= 5$), la vitesse de translation est nulle. Le robot peut changer de direction mais ne peut avancer en direction de l'obstacle.

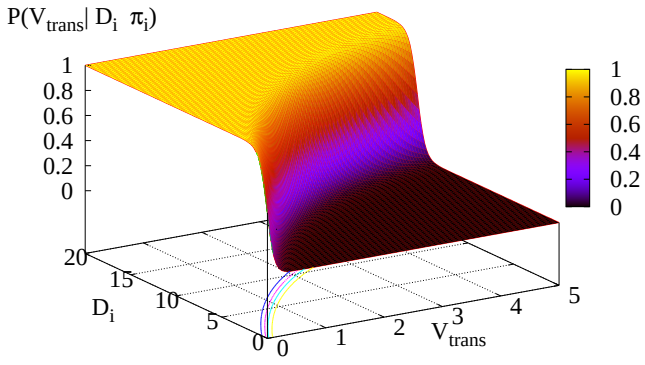
4.2.2 Le sous-modèle "'suivi de consigne"'

L'identification des paramètres du sous-modèle de suivi de consigne consiste à régler l'écart-type des gaussiennes suivantes :

$$P(V_t \mid V_{td}) = G_{(\mu=V_{td},\sigma_t)}$$
$$P(V_r \mid V_{rd}) = G_{(\mu=V_{rd},\sigma_r)}$$

$$f_{trans}(V_t) = D_{max}(1 - \frac{1}{2k}e^{-0.20\frac{V_t}{k}})$$

$$k = 0.5$$



Tab. 4.2 : Évitement d'obstacles : f(Vtrans, Distance)

Dans un premier temps, les σ sont fixés a priori. Pour cela rappelons que le modèle suivi de consigne ne doit interdire aucune commande, mais seulement désigner celle souhaitée. Ainsi, VT_{min} doit être autorisé même si la consigne était VT_{max}.

À $\mu \pm 3\sigma$, la valeur de probabilité $P()$ est de l'ordre de 1% de sa valeur maximale (cf. figure 4.5). Nous fixons donc les σ de façon à ce que les intervalles de valeurs de Vt et Vr soient égaux à $3 \times \sigma$. On obtient alors :

$$\sigma_t = \frac{Vt_{max} - Vt_{min}}{3}$$

$$\sigma_r = \frac{Vr_{max} - Vr_{min}}{3}$$

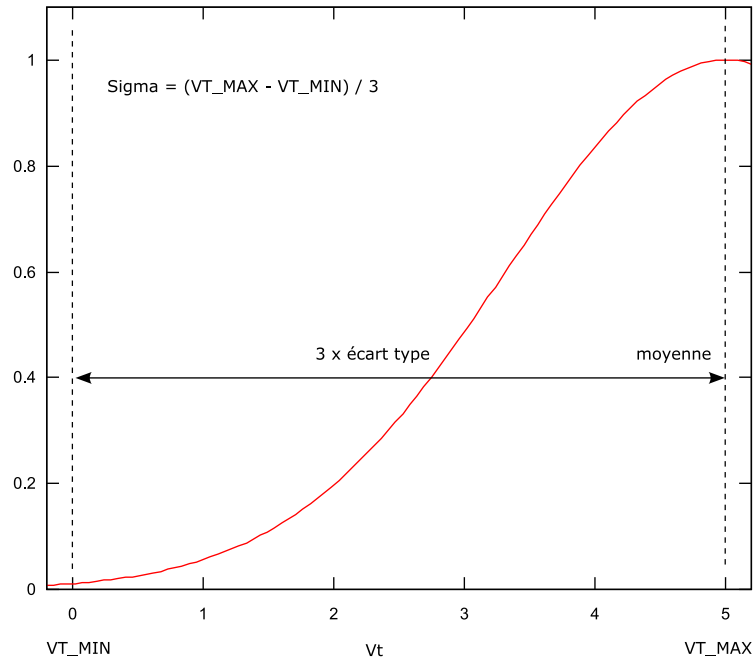


Fig. 4.5 : Modèle suivi de consigne : identification des paramètres

4.3 Utilisation

Nous sommes arrivés au terme des phases de spécification et d'identification. Nous disposons maintenant d'un programme bayésien d'évitement d'obstacles complètement défini. La phase d'utilisation va consister à poser en boucle la question suivante :

$$P(V_t\ V_r \mid V_{td}\ V_{rd}\ D1 \ldots D8)$$

V_t et V_r sont les commandes motrices à envoyer directement au contrôleur de bas niveau. V_{td} et V_{rd} sont les commandes désirées fournies par le contrôleur de comportement. $D1 \ldots D8$ sont les variables d'observations dont les valeurs sont issues du prétraitement des données sensorielles que nous décrivons dans la section suivante.

Pré-traitement des données sensorielles

Nous utilisons comme capteur le télémètre laser positionné à l'avant du robot. Il balaye de droite à gauche sur 180° avec une résolution angulaire de 0.5°. Nous disposons donc d'un tableau de 361 mesures. Huit est le nombre de zones retenues. Chaque zone a donc une taille angulaire de 22.5°, et traite 45 mesures. Nous affectons à D_i la plus petite valeur des 45 mesures. Dans le cas où plusieurs obstacles se trouvent dans la même zone, nous retenons ainsi la distance de celui le plus proche.

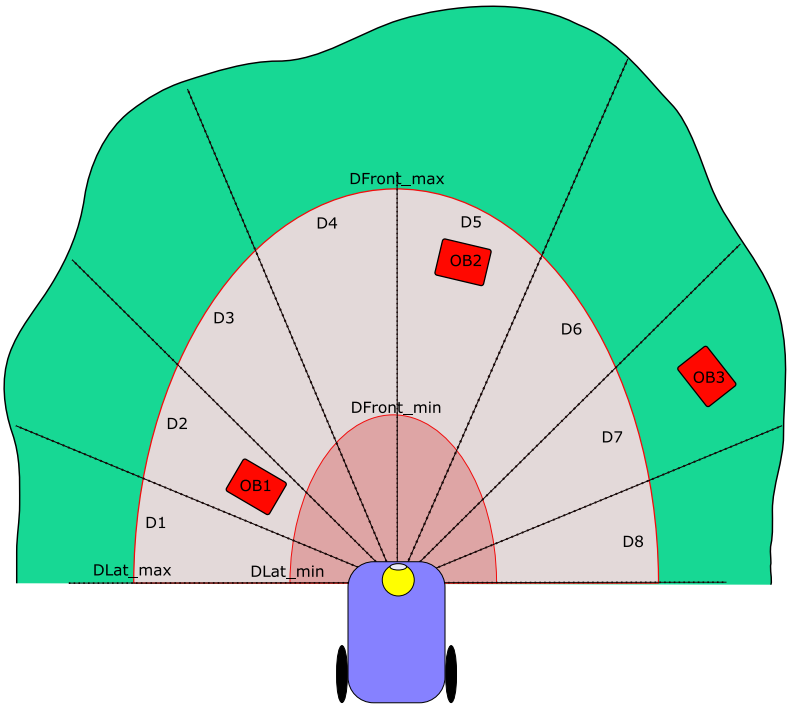
$$D_i = min(m_{i1} \ldots m_{i45})$$

4.4 Résultats de l'évitement d'obstacles réactif

Pour illustrer le fonctionnement du programme bayésien d'évitement d'obstacles, on se place dans les conditions suivantes :

- Les commandes désirées correspondent à une vitesse de translation maximum, $Vt = 5$, et une vitesse de rotation $Vr = 0$.
- Auncun obstacle n'est présent en D1, D3, D4, D6, D8.
- Un obstacle est très proche en D2, proche en D5 et éloigné en D7.

La figure 4.3 illustre cette situation.

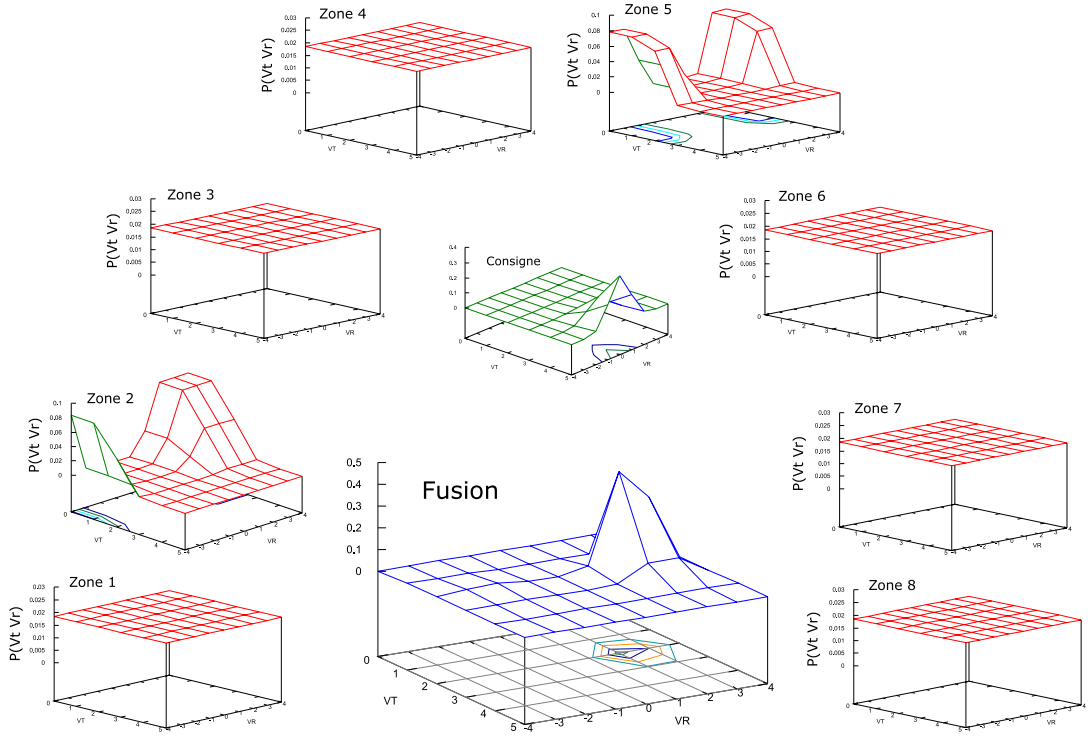


Tab. 4.3 : Évitement d'obstacles : situation expérimentale

La figure 4.4 montre les résultats de l'exécution du programme bayésien d'évitement d'obstacles. La distribution du sous-modèle suivi de consigne est en vert au centre de la figure. Nous observons bien un pic de probabilités pour les comandes désirées ($Vt = 5$ et $Vr = 0$).

Les distributions des sous-modèle proscriptifs sont positionnées suivant l'orientation des zones sur le robot. Nous remarquons que les zones n'ayant pas détectées d'obstacles n'interdisent ni ne favorisent de commandes motrices. Ceci est également vrai pour la zone $D7$, car l'obstacle est à une distance supérieure à D_{max}. $D2$ a détecter l'obstacle le plus proche. Elle interdit les vitesses de translation supérieures à 2, et permet les rotations sur la droite. $D5$ interdit au robot d'aller tout droit à des vitesses de translations supérieures à 3.

Le résultat de la fusion est la distribution en bleu au centre de la figure. Comme nous l'espérions le robot va s'orienter vers la droite ($Vr = 2$) et réduire sa vitesse ($Vt = 2$) pour éviter l'obstacle *OB1*.



Tab. 4.4 : Évitement d'obstacles : résultat de la fusion des sous-modèles

4.5 Conclusion

Nous disposons maintenant d'un module d'évitement d'obstacles capable de sécuriser les mouvements du robot. Nous pouvons noter la simplicité et la clarté de la décomposition en sous-problèmes simples dont les résultats sont ensuite fusionnés. Notons également la souplesse de la liaison avec le contrôleur de comportements : il peut demander de tourner à droite, exactement à 1 m/s (consigne prescriptive) ou légèrement à gauche sans dépasser 0.5 m/s (consigne proscriptive). Ce type de consigne s'exprime facilement comme distribution de probabilités.

Les résultats du module d'évitement d'obstacles seront présentés sous forme de séquence de photos au chapitre « Résultats ».

Chapitre 5

Implémentation d'un comportement animal sur le robot BIBA

Dans les deux chapitres précédents nous avons présenté les programmes bayésiens du contrôleur de comportements et de la fonction d'évitement d'obstacles.

Ce chapitre présente l'implémentation de ces modules sur le robot BIBA, ainsi que les librairies de prétraitement des données sensorielles et des commandes motrices.

5.1 Un système distribué

Comme illustré par la figure 5.1, trois calculateurs se répartissent les trois tâches principales de l'expérience : la gestion de bas niveau des données capteurs et commandes motrices, le traitement vidéo et le contrôle sécurisé des comportements du robot. Le support de communication est un protocole HTTP/CGI.

Sur un système d'exploitation temps réel, *XO/2*, le PowerPC (*bibabot*) exécute une couche d'abstraction de bas niveau. Ainsi, les données sensorielles et les commandes motrices sont accessibles via une interface HTTP/CGI. Il intègre également un module de sécurité basique capable de débrayer les moteurs lorsque les "bumpers" du robot entrent en contact avec un obstacle.

Le portable (*bibabotp*), seul à posséder une entrée firewire (IEEE1394), est en charge de la capture et du traitement des données massives de la caméra. Ce sera son unique rôle car ces tâches sont gourmandes en ressources CPU et mémoire. Dans un souci d'avoir un protocole de communication homogène pour toute l'application, un serveur web a été installé sur *Bibabotp* (ie Apache). De cette façon, les données sensorielles issues du traitement vidéo sont également disponibles par le biais de requêtes HTTP/CGI. *bibabotp* est également le serveur du fichier de démarrage pour le PowerPC.

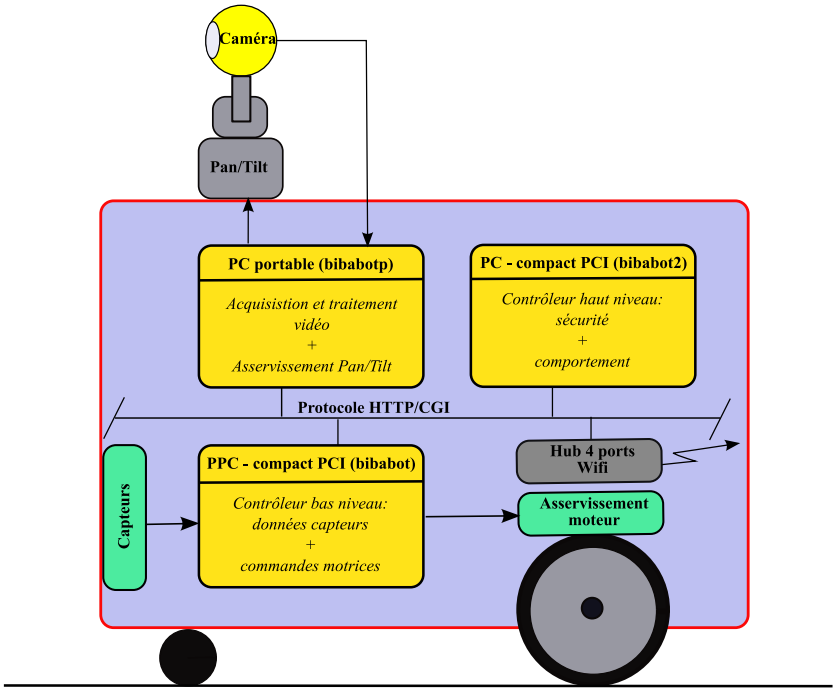


Fig. 5.1 : Distribution des tâches

Enfin, *bibabot2* joue le rôle de contrôleur de haut niveau. Il est responsable du comportement général du robot. A chaque pas de temps, il évalue le nouveau comportement à adopter en accord avec les données capteurs fournies par les deux précédents calculateurs. Grâce à son module d'évitement d'obstacles, il garantit également que les commandes motrices envoyées au système d'asservissement soient sûres.

5.2 Les modules du système

Pour l'exécution des tâches listées ci-dessus nous avons implémenté les modules suivants, également représentés dans la figure 5.2 :

- *BBehaviorCtrl* est le contrôleur de comportements présenté au chapitre 3. Son implémentation est détaillée en section 5.6.
- *BBotOBA* est le module d'évitement d'obstacles présenté au chapitre 4. Nous détaillons son implémentation en section 5.5.
- *BBot* est une couche d'abstraction du matériel de deuxième niveau. L'implémentation de ce module est présentée en section 5.4.
- BBCam et CamCGI sont les modules pour l'acquisition et le traitement du flux vidéo de la caméra. Ils seront explicités en section 5.7.

5.3 Outils utilisés

Les différents modules listés ci-dessus ont été développés principalement avec des outils dits Open Source, d'une part, pour leur gratuité, et d'autre part, pour leur

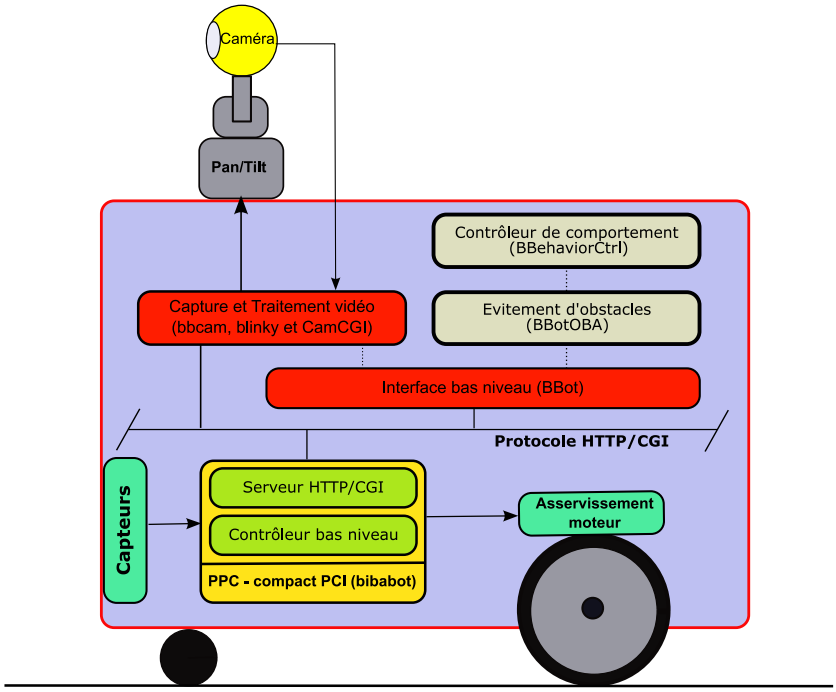


Fig. 5.2 : Diagramme de l'architecture

efficacité reconnue. Ils ont souvent l'avantage d'être multi-plates-formes. Nous listons ci-dessous les principaux outils utilisés.

L'environnement de développement que nous avons choisi est *Eclipse* (voir présentation dans [Ecl]). C'est un outil convivial et ces "wizards" le rendent intuitif. Mais *Eclipse* a surtout l'avantage d'être extensible. Même si le langage de programmation natif est JAVA, une extension, nommé **CDT**[1], permet de développer en C/C++. *Eclipse* dispose également d'une interface graphique de débogage capable de s'interfacer avec GDB[2] (cf. documentation dans [GDB06]), un débogueur largement utilisé. *Eclipse* intègre un client CVS[3] avec une fonction de comparaison entre un fichier ou un répertoire local avec celui sur le serveur.

L'implémentation du système de traitement d'images est basé sur la librairie *openCV* (version v0.9.6), proposée par Intel. La documentation et les exemples sont abondants. La documentation peut être consultée dans [Int03].

ProBT©, présentée en section 1.7, est utilisé pour l'implémentation des programmes bayésiens. Cette librairie est donc utilisée pour développer le contrôleur de comportements et le module d'évitement d'obstacles.

Finalement, nous avons utilisé *Gnuplot* (cf. documentation dans [Gnu04]), un outil de tracé de courbes, principalement pour analyser les distributions de probabilités issues des programmes bayésiens. Il a été très utile pour le réglage des paramètres des formes paramétriques.

[1] C/C++ Development Tools – voir manuel de référence dans [CDT05]
[2] GNU Project Debugger
[3] Concurrent Versions System – http ://ximbiot.com/cvs/

5.3.1 Configuration du système

L'entière application est paramétrée par deux fichiers de configuration : un pour le système d'acquisition et de traitement vidéo, l'autre pour le contrôleur de comportements. Nous avons fait le choix d'utiliser le format XML pour la description de ces fichiers. C'est un format flexible permettant d'étendre les paramètres de l'application facilement. De plus, autour de ce format, plusieurs outils ont été développés pour pouvoir le traiter. Nous avons choisi la librairie *libXML2* comme parser (cf. documentation dans [Gno]), car elle est gratuite, largement utilisée, et bénéficie d'une communauté de développement dynamique.

Nous avons construit une classe générique, nommée *CCfgMgr*, pour interfacer les fichiers de configurations avec le reste du code. Cette classe est illustrée sur la figure 5.3. Le tableau 5.1 recense les méthodes publiques de cette classe.

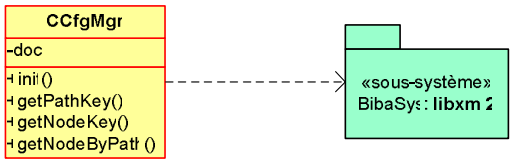


Fig. 5.3 : La classe d'interface pour les fichiers de configuration

Méthodes	Description
init()	Charge le fichier XML et crée un pointer sur l'objet racine
getPathKey(path, key, prop)	Retourne la valeur de l'objet *path + Key*, ou de l'attribut *prop* si non null.
getNodeKey(node, key, prop)	Retourne la valeur de l'objet *Node*, ou de l'attribut *prop* si non nul.
getNodeByPath(path)	Retourne l'objet identifié par *path*

Tab. 5.1 : Accessoires ajoutés au robot

5.4 Interface de bas niveau : *BBot*

Le module *BBot* peut être vu comme une couche d'abstraction de matériel de deuxième niveau, pour le module d'évitement d'obstacles et le contrôleur de comportements.

5.4.1 Spécifications

Au chapitre 2 nous avons présenté le contrôleur de bas niveau embarqué sur le robot BIBA, plus exactement sur *bibabot*, le PowerPC. Il sert principalement d'interface pour l'accès aux données sensorielles et aux commandes motrices via un protocole HTTP/CGI.

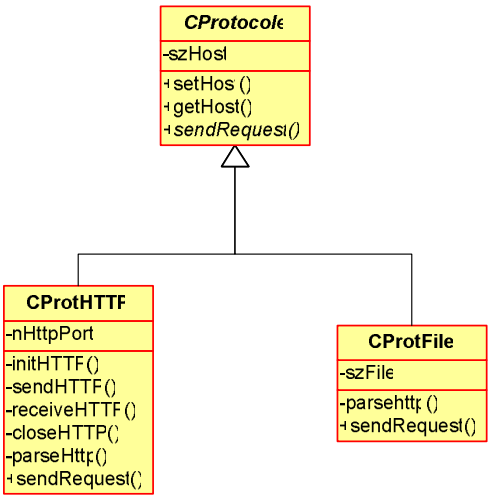


Fig. 5.4 : Implémentation de la classe générique *CProtocole*

Le module *BBot* doit proposer l'accès aux mêmes informations et actions, mais via l'interface de classes. De cette façon, le protocole de communication est masqué, ainsi que le format des données échangées.

Le contrôleur de bas niveau ne propose pas de mécanismes d'évènements, or il est important d'être notifié quand le niveau de la batterie est trop faible ou si les moteurs sont débrayés parce que les plaques tactiles du robot ont touchés un obstacle. La librairie *BBot* doit donc implémenter un mécanisme d'évènements.

5.4.2 Protocole de communication

Le protocole de communication avec le contrôleur de bas niveau est imposé par *Bluebotics*, le constructeur du robot. L'interface d'accès aux capteurs et aux moteurs est en réalité une collection de composants CGI gérés par un serveur web. Le protocole ainsi utilisé n'est donc autre que du HTTP[4].

Pour éviter d'être connecté en permanence sur le robot pendant la phase d'implémentation, nous avons utilisé un protocole qui simule les réponses du robot. Pour cela nous avons créé une collection de fichiers réponses au format HTML pour chaque commande. Ces fichiers ont été créés via un navigateur internet, en envoyant une à une les requêtes au contrôleur de bas niveau, et en sauvegardant les réponses.

D'autres protocoles sont envisageables : *Bluebotics* peut, dans une nouvelle version du logiciel embarqué, changer le format des réponses, voire même, changer de type de protocole.

Pour minimiser l'impact d'un changement de protocole sur le code existant, nous proposons de créer une classe générique *CProtocole* avec une méthode virtuelle *sendRequest()*. De cette façon, le protocole réellement utilisé est masqué aux classes utilisatrices de *CProtocole*. Le schéma UML 5.4 illustre l'implémentation des protocoles HTTP et « fichiers ».

[4]Hypertext Transfer Protocol

La méthode *sendRequest* est déclarée virtuelle. Chaque nouveau protocole dérivant de *CProtocole* doit l'implémenter.

```
virtual int sendRequest(char* szRequest, char** szResponse)=0;
```

5.4.3 Mécanisme de gestion d'évènements

Le mécanisme de gestion d'évènements tel que nous l'avons implémenté est représenté dans la figure 5.5. Ce mécanisme est proche de celui proposé par le langage JAVA. On retrouve la notion de "Listener" (écouteurs). Un écouteur est un objet qui doit être notifié d'un ou plusieurs événements. Dans cet objectif, il doit, d'une part, s'enregistrer auprès des objets générateurs d'évènements, et d'autre part, posséder au moins une méthode qui pourra être invoquée si l'évènement attendu apparaît. En d'autres termes, l'écouteur doit remplir un contrat bien particulier : dans notre proposition il doit dériver d'une classe de type *CEventListener*.

En dérivant de la classe *CBibaBotEventListener*, l'écouteur pourra être notifié des évènements quand le niveau de la batterie est trop faible, ou si les moteurs sont débrayés parce que les plaques tactiles du robot ont touché un obstacle. Un écouteur peut également recevoir les actions de l'utilisateur sur le "joystick" en dérivant de la classe *CJoystickEventListener*.

Comme nous l'avons vu plus haut, un écouteur doit s'enregistrer auprès d'objets générateurs d'évènements. Un écouteur peut écouter plusieurs sources d'évènements et une source d'évènements peut alerter plusieurs écouteurs. Il est donc nécessaire que les générateurs proposent un mécanisme d'enregistrement multi-écouteurs. Ce mécanisme est disponible dans la classe *CEventSender*. Elle conserve une liste de références sur les écouteurs qui se sont enregistrés via la méthode *addListener()*. Ainsi, les classes d'objets sources d'évènements doivent dériver de *CEventSender* pour hériter de ce mécanisme.

5.4.4 Implémentation

Cette interface de bas niveau est construite sous forme d'une librairie dynamique, nommée *libBBot*. Le diagramme de classes représenté sur la figure 5.6, recense ces principales classes.

La classe *CBibabotbase* est composée de tous les modules gérés par le contrôleur de bas niveau : les capteurs et les contrôleurs.

CBibabot étend la classe *CBibabotbase*, en ajoutant la gestion de la caméra et du joystick. Le joystick est utilisé pour travailler en mode simulation (cf. section 5.4.6). Dans ce mode, les observations de la caméra sont simulées en utilisant le joystick. Ceci a permis de travailler en parallèle sur le comportement du robot et sur le système d'acquisition et de traitement d'image. Afin d'obtenir les évènements du joystick, *CBibabot* dérive de classe *CJoystickEventListener* et réimplémente les méthodes correspondants aux évènements qu'elle doit traiter.

Elle est également la source des évènements liés à la batterie et aux plaques tactiles. Pour cela elle dérive de classe *CEventSender* pour hériter du mécanisme

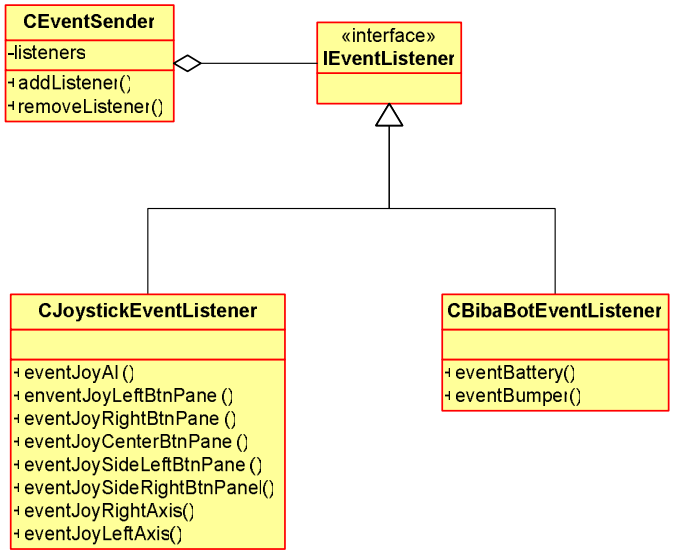


Fig. 5.5 : Diagramme de classes du mécanisme d'événements

d'enregistrement de "Listener" (cf. section 5.4.3). Pour pouvoir générer les évènements, elle initialise un "thread" qui toutes les 20 millisecondes, envoi des requêtes au contrôleur de bas niveau pour connaître l'état de la batterie et des plaques tactiles. Le "thread" invoque les méthodes *fireEventBattery()* et *fireEventBumper()* qui parcourent un par un les écouteurs enregistrés de l'évènement correspondant.

CBibabot initialise également un thread pour les commandes motrices. Toutes les 20 millisecondes, il envoi au contrôleur de bas niveau les nouvelles commandes motrices. Ces commandes sont le résultats d'un traitement (cf. section 5.4.5) sur les ordres moteurs reçus via la méthode *setMotorAction()*.

Comme nous l'avons vu plus haut, la classe *CBibabotbase* est composée de tous les modules gérés par le contrôleur de bas niveau : Les capteurs et les contrôleurs. À chaque module nous avons associé une classe, qui a pour objectif de fournir des méthodes d'accès aux données du module tout en masquant le protocole utilisé et le format des données.

Ceci a motivé une implémentation selon le diagramme de classes présenté sur la figure 5.7

5.4.5 Traitement des commandes motrices

Les commandes motrices avant traitement, sont constituées d'une vitesse de translation ($Vtrans$) et d'une vitesse de rotation ($Vrot$). Le traitement se décompose en trois étapes : limitation de force centrifuge, limitation des accélérations et transformation des vitesses de rotation et translation en vitesse roue gauche et roue droite.

Pour limiter la force centrifuge, deux actions sont possibles : réduire la vitesse de rotation lorsque la vitesse de translation et grande, ou inversement, limiter la vitesse de translation si la vitesse de rotation est importante. Afin d'avoir un déplacement sûr, la deuxième option est plus judicieuse. Par exemple, dans une manoeuvre d'évitement d'un obstacle, le changement de direction est plus important que de conserver

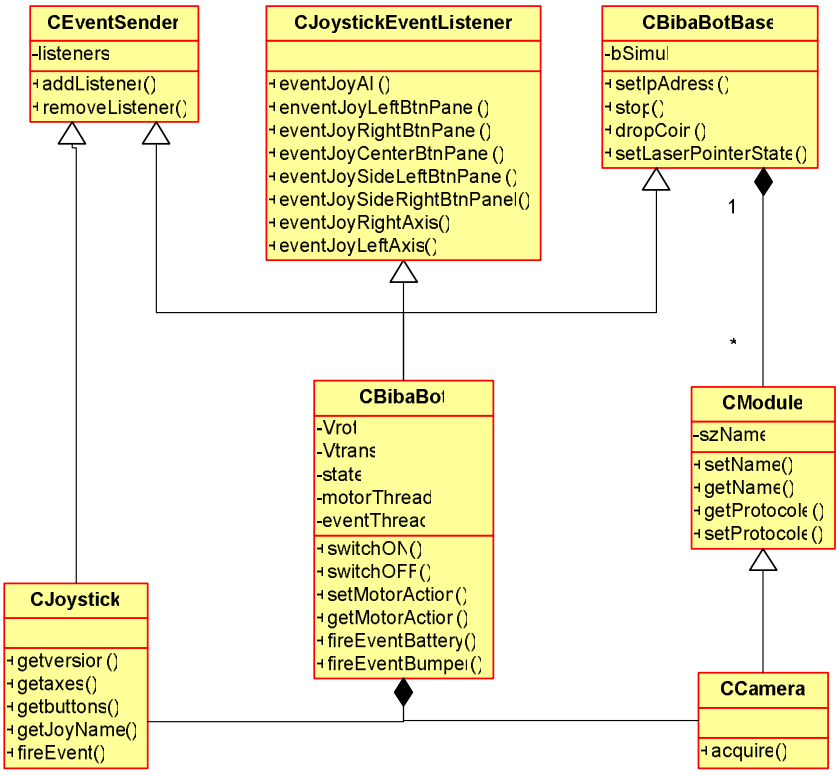


Fig. 5.6 : Diagramme de classes du module *BBot*

la vitesse de translation. Pour implémenter ce comportement, nous utilisons la sigmoïde illustrée sur la courbe de gauche de la figure 5.8. La courbe correspond à la valeur maximale de la vitesse de translation connaissant la vitesse de rotation. Cette étape assure que les commandes motrices restent sur ou en dessous de la courbe.

La deuxième étape consiste à limiter l'accélération du robot, afin d'éviter les changements brusques de vitesse. Pour cela on applique un filtre passe-bas aux commandes motrices. L'équation du filtre passe-bas appliquée à la vitesse de translation est donnée ci-dessous :

$$VT_{final} = \alpha V_{trans} + (1 - \alpha) V_{trans}^{t-1}$$

Comme illustré sur le graphe de droite de la figure 5.8, la vitesse issue de filtre, en rouge, tend progressivement vers la commande désirée tout en lissant les angles.

Les ordres moteurs attendus par le contrôleur de bas niveau, sont constitués de deux vitesses de rotations : une pour la roue droite (Vd) et l'autre pour la roue gauche (Vg). La dernière étape consiste à convertir les vitesses de translation et rotation, en appliquant les équations suivantes :

$$Vd = V_{rot} - V_{trans} \tag{5.1}$$
$$Vg = V_{rot} + V_{trans} \tag{5.2}$$

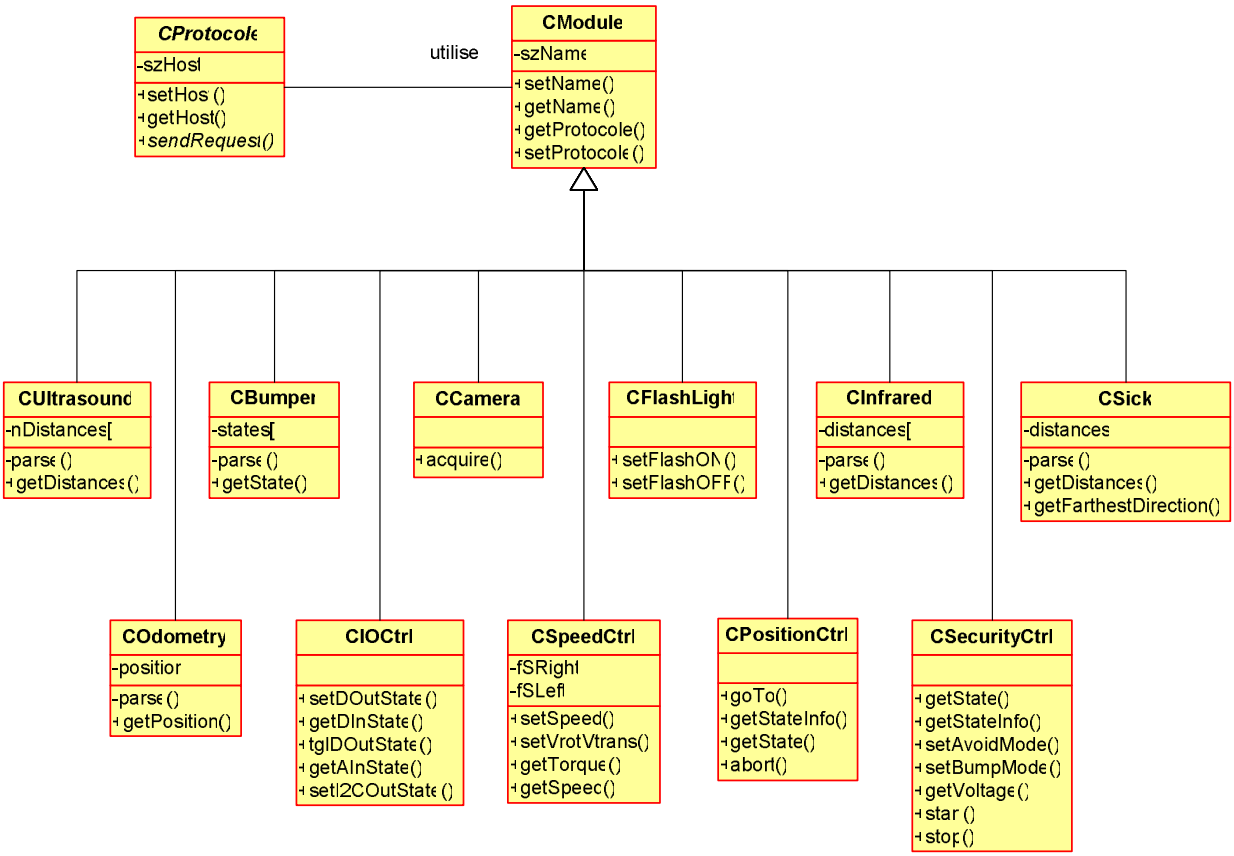


Fig. 5.7 : Implémentation de la classe générique *CModule*

5.4.6 Simulateur des données sensorielles

Nous avons crée un mode simulation pour pouvoir implémenter les comportements sans devoir être connectés au robot. Le constructeur de la classe *CBibaBotBase* accepte un paramètre *bSimul*. Lorsqu'il est positionné à vrai, les classes des modules n'utilisent plus le protocole *CProtHTTP* mais *CProtFile*. Les mesures du télémètre laser sont alors lus dans un fichier (cf. section 5.4.2).

Les observations de la caméra sont simulées en utilisant le joystick. Trois des quatre boutons frontaux sont utilisés pour valider la présence du prédateur, du maître et de la proie. La direction et la distance sont simulées par les axes du joystick.

Le mode simulation a également permis de travailler sur le comportement du robot, alors que le système d'acquisition et de traitement d'image était en cours de développement.

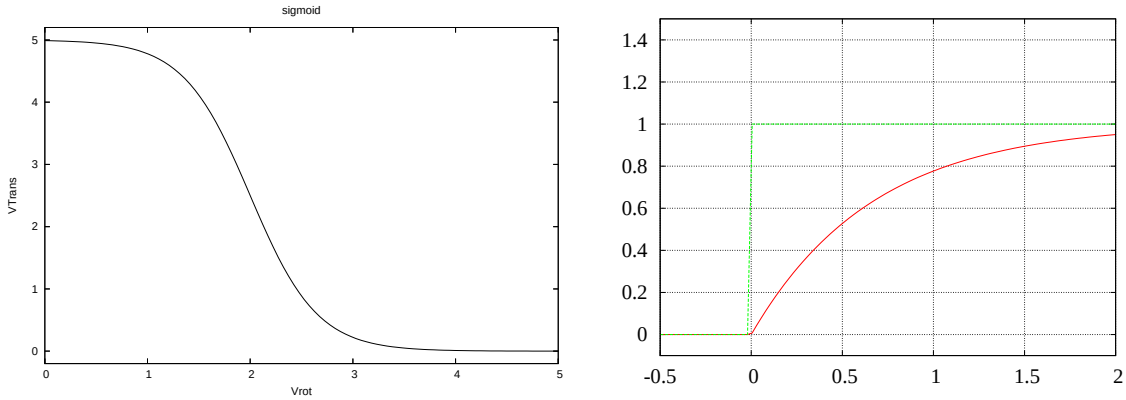


Fig. 5.8 : Traitement des commandes motrices : à gauche, limitation de la force centrifuge ; à droite, limitation des accélérations

5.5 Évitement d'obstacles : *BBotOBA*

Le rôle du module d'évitement d'obstacles, que nous avons nommé *BBotOBA*, est de transmettre au système d'asservissement de bas niveau, autant que possible, les commandes choisies par le contrôleur de comportement, tout en garantissant la sécurité du robot et de son environnement.

5.5.1 Spécifications

Pour ce faire, *BBotOBA* doit implémenter le programme bayésien d'évitement d'obstacles présenté au chapitre 4. Il doit définir les sous-modèles proscriptifs associés à chaque zone, ainsi que le sous-modèle suivi de consigne. Il doit pouvoir recevoir les commandes motrices désirées du contrôleur de comportements, les transmettre au programme d'évitement d'obstacles et envoyer les ordres moteurs résultants au contrôleur de bas niveau.

Dans le cas où le contrôleur de comportements impose l'immobilité, le robot n'évite pas les obstacles dynamiques qui passeraient près de lui. Cela signifie que si V_{td} **et** V_{rd} sont nulles, le module d'évitement d'obstacles n'est pas sollicité.

5.5.2 Implémentation

Ce module est construit sous forme d'une librairie dynamique, nommée *libBBotOBA*. Nous implémentons la fonction d'évitement d'obstacles suivant le schéma illustré figure 5.9.

Le sous-modèle proscriptif est défini dans la classe *CZone*. La classe *CSickObsAvoid* implémente le sous-modèle suivi de consigne, ainsi que la fusion des sous-modèles.

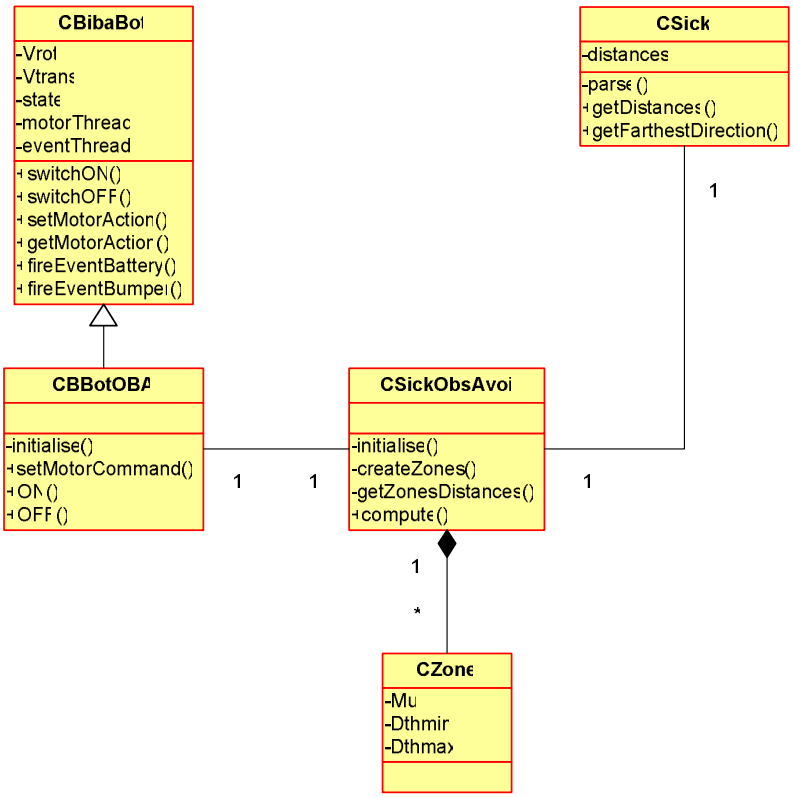


Fig. 5.9 : Diagramme de classes du module d'évitement d'obstacles

5.5.3 Création des zones

Le module d'évitement d'obstacles a été implémenté de façon à automatiser au maximum la création des zones tout en minimisant sa configuration de la part de l'utilisateur. Rappelons que chaque zone attend les paramètres suivants :

- ses seuils de contraintes D_{min} et D_{max} (cf. 4.2.1),
- et sa position relative au robot μ (cf. 4.2.1).

Voyons maintenant comment nous avons automatisé le calcul des paramètres.

Calcul des seuils de contraintes : D_{min} et D_{max}

Le choix des seuils de contraintes dépend fortement de l'application dans laquelle est intégrée le module d'évitement d'obstacles. Dans notre cas le robot doit pouvoir :

- En ligne droite, se déplacer à la vitesse maximum autorisée pour cette expérience. Par contre, plus la vitesse de braquage est importante, plus la vitesse de translation est faible, jusqu'à être nulle lorsque le robot fait un demi-tour. Il est ainsi naturel d'avoir une distance de sécurité plus importante à l'avant du robot que sur ses côtés ($DFront_{max}$ assez grande et supérieure $DLat_{max}$).
- Se déplacer en ligne droite dans un couloir de largeur moyenne ($1,50m$) sans être contraint sur la vitesse de translation ($DLat_{max}$ assez petite).
- De plus, il doit pouvoir faire demi-tour dans ce couloir ($Dfront_{min}$ assez petite).

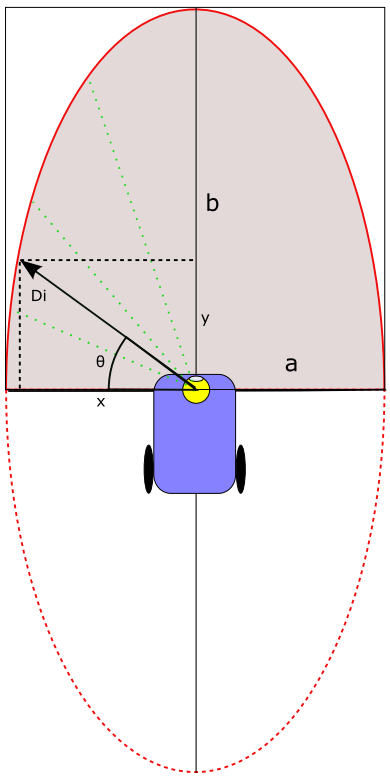
Ces contraintes nous amènent à décrire deux ellipses autour du robot illustrées par la figure 4.4. L'ellipse la plus proche définit les Dk_{min}, et la plus éloignée les Dk_{max}. Les seuls paramètres à régler par l'utilisateur sont les distances frontales ($DFront_{min}$, $DFront_{max}$) et latérales ($DLat_{min}$, $DLat_{max}$) de contraintes.

Les distances D_{min} et D_{max} peuvent alors être calculées à partir du point d'intersection des ellipses et de la médiane de la zone (cf 5.2), d'où l'équation :

$$D_i = \frac{ab}{\sqrt{(b\cos\theta)^2 + (a\sin\theta)^2}} \tag{5.3}$$

Avec :

- a : distance latérale de contrainte, donnée par l'utilisateur,
- b : distance frontale de contrainte, donnée par l'utilisateur,
- θ : angle d'orientation de la zone, déduit du nombre de zone.



D_i peut être déduit de y :

$$D_i = \frac{y}{sin\theta} \tag{5.4}$$

y peut à son tour être calculé à partir de l'équation de l'ellipse sans rotation :

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{5.5}$$

x peut être remplacé par :

$$x = \frac{y}{tan\theta} \tag{5.6}$$

En remplaçant 5.6 dans 5.5 et en isolant y on obtient :

$$y = \frac{ab\tan\theta}{\sqrt{b^2 + (a\tan\theta)^2}} \tag{5.7}$$

Finalement, en remplaçant 5.4 dans 5.7 :

$$D_i = \frac{ab}{\cos\theta\sqrt{b^2 + (a\tan\theta)^2}} \tag{5.8}$$

Qui s'écrit également :

$$D_i = \frac{ab}{\sqrt{(b\cos\theta)^2 + (a\sin\theta)^2}} \tag{5.9}$$

Tab. 5.2 : Calcul des seuils de contraintes

Calcul de la position des zones : μ

μ est la position angulaire de la zone relative au robot. L'angle 0 est droit devant le robot. μ est déduit de la couverture du capteur, dans notre cas 180°, et du nombre de zones.

5.5.4 Configuration du module

Nous entendons par configuration du module d'évitement d'obstacles, le choix du nombre de zones, la définition des seuils de contraintes utilisés par les sous-modèles proscriptifs et le réglage des paramètres des formes paramétriques du sous-modèle suivi de consigne.
La configuration se fait entièrement dans le fichier *SICKOBSAVOID.H* (cf. figure 5.3).

```
...
#include "zone.h"
#include "libbbot.h"

#define NB_ZONE          8
#define SICK_RANGE       180

#define DIST_SIDE_MIN    550
#define DIST_SIDE_MAX    800
#define DIST_FRONT_MIN   550
#define DIST_FRONT_MAX   1000

#define VRD_STDDEV       (VR_MAX - VR_MIN) / 3
#define VTD_STDDEV       (VT_MAX - VT_MIN) / 3
...
```

Tab. 5.3 : Calcul des seuils de contraintes

5.6 Contrôleur de comportements : *BBehaviorCtrl*

5.6.1 Spécifications

Le rôle du contrôleur de comportements, que nous avons nommé *BBehaviorCtrl*, est de choisir, à chaque pas de temps, un comportement approprié basé sur les observations de son environnement et de son "vécu". À partir du comportement choisi, il doit proposer les commandes motrices appropriées au module d'évitement d'obstacles.

Pour cela, le contrôleur doit implémenter et mettre en oeuvre les quatre filtres élémentaires décrits au chapitre 3 : le filtre prédateur, maître, proie et route de fuite.

5.6.2 Configuration du contrôleur

Le contrôleur de comportements est configuré par le fichier *config.xml*. La figure 5.4 en propose un extrait. Il est composé de trois sections principales : *bot*, *directories* et *objects*.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<config>
  <!--****************************************
   system settings
  *****************************************-->
  <bot>
    <ppc>
      <host>194.199.21.27</host>
      <battlevs>
        <warning>23</warning>
        <critical>22</critical>
      </battlevs>
    </ppc>
    <camera>
      <host>194.199.21.203</host>
    </camera>
  </bot>
  <!--****************************************
     Objects settings
  *****************************************-->
  <directories>
    <modelsdatadir>data</modelsdatadir>
  </directories>
  <objects>
  <Predator>
   <DynModelFilter>PredDynTable.dat</DynModelFilter>
   <SenModelFilter>SensorModel.dat</SenModelFilter>
   <BhrModelFilter>BHPredTable.dat</BhrModelFilter>
   <odour enabled="1">
   <duration>15</duration>
   </odour>
   <camera enabled="1">
     <color>RED</color>
   </camera>
  </Predator>
  ...
  </objects>
</config>
```

Tab. 5.4 : Fichier de configuration du contrôleur de comportements

La section *bot* contient les adresses IPs du PowerPC (*bibabot*, le contrôleur de bas niveau) et du serveur traitant les données de la caméra (*bibabotp*). Ces adresses sont les attributs "*szHost*" des objets issus de *CProtHTTP* (cf. 5.4.2). Le contrôleur vérifie régulièrement le niveau des batteries. L'objet XML *battlevs* définit deux seuils : *warning* et*critical*. Le contrôleur avertit l'utilisateur lorsque le niveau de la batterie passe en dessous du seuil *warning*, et stoppe le robot lorsque le niveau atteint le seuil *critical*.

Le paramètre *modelsdatadir* dans la section *directories* donne l'emplacement des fichiers d'initialisation des modèles.

La section *objects* est composée d'un groupe de paramètres par centre d'intérêts avec qui le robot interagit : le maître, le prédateur, la proie et la route de fuite. L'extrait de la figure 5.4 donne à titre d'exemple la description de l'objet prédateur. Le nom des fichiers d'initialisation des modèles dynamiques, capteur et de comportement sont respectivement les paramètres *DynModelFilter*, *SenModelFilter*, *BhrModelFilter*.

Les filtres qui utilisent la caméra comme capteur, doivent définir la couleur du sujet, les valeurs possibles sont : RED, BLUE et GREEN. L'utilisation de la caméra peut être désactivée, auquel cas la présence, la distance et la direction du sujet peuvent être simulées en utilisant le joystick (cf. section 5.4.6).

5.6.3 Implémentation

Ce module est construit sous forme d'une librairie dynamique, nommée *libBBehaviorCtrl*. Nous implémentons le contrôleur de comportements suivant le diagramme de classes illustré figure 5.10.

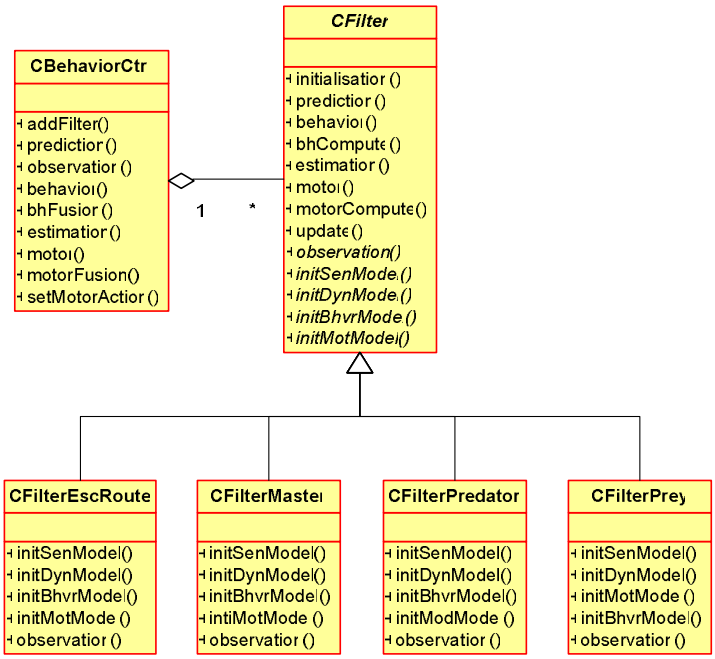


Fig. 5.10 : Diagramme de classes du contrôleur de comportements

5.6.4 Le filtre élémentaire : *CFilter*

Le robot est contrôlé par quatre filtres sensori-moteurs élémentaires (décrits en section 1.6) : prédateur, proie, maître et route de fuite. La spécification des quatre filtres est à quelques variantes près similaire. La classe *CFilter* définit ainsi le canevas du filtre élémentaire. Elle déclare les variables locales au filtre et définit la distribution conjointe et sa décomposition.

On retrouve également dans *CFilter*, la déclaration des quatre modèles : dynamique, capteur, comportement et moteur. Toutefois, le choix des formes paramétriques et l'identification des paramètres sont propres à chaque filtre. Par conséquent, *CFilter* déclare quatre méthodes virtuelles pour l'initialisation des modèles : *initSenModel()*, *initDynModel()*, *initBhvrModel()* et *initMotModel()*. Ces méthodes sont implémentées dans *CFilterMaster*, *CFilterPredateur*, *CFilterPrey* et *CFilterEscRoute*.

La méthode *observation()* est le point d'appel pour exécuter l'acquisition et le prétraitement des données sensorielles, dans le but de mettre à jour les variables d'observation. Cette tâche est spécifique à chaque filtre : par exemple, le filtre route de fuite utilise le télémètre laser comme capteur, alors que les autres filtres utilisent la caméra. Par conséquent, cette méthode est déclarée virtuelle et est implémentée par chaque classe spécialisée.

Le filtre doit également répondre aux quatre questions probabilistes : prédiction, comportement, estimation et moteur. Pour cela, *CFilter* implémente respectivement les fonctions *prediction()*, *behavior()*, *estimation()* et *motor()* pour chacune des questions. Les méthodes *bhCompute()* et *motorCompute()* construisent les distributions de probabilités pour la fusion du comportement et des commandes motrices.

Enfin, la méthode *update()* prépare le filtre pour le pas de temps suivant.

5.6.5 Paramètres des modèles

Les formes paramétriques des modèles sont des tables de probabilités définies au chapitre 3. Certaines d'entre elles sont entièrement définies dans le code du contrôleur. C'est la cas lorsque les valeurs de probabilités sont obtenues à partir d'équations mathématiques. Par exemple, le modèle moteur du filtre prédateur interdit au robot de se déplacer dans la direction où se trouve le prédateur, si le comportement choisi est la fuite. Le prédateur est assimilé, dans ce cas, à un obstacle. Par conséquent, les valeurs de probabilités sont calculées à partir des équations d'évitement d'obstacles données en section 4.2.1. Ainsi, les tables du modèle moteur du filtre prédateur sont définies dans sa méthode *initMotModel()*.

Pour les cas où les valeurs sont données "manuellement", typiquement pour le modèle de comportement, nous avons mis en place un mécanisme de définition de table par le biais d'un fichier texte. Cette méthode est utilisée autant que possible, car elle présente une plus grande souplesse : il n'est pas nécessaire de compiler le code à chaque changement de paramètre.

5.6.5.1 Initialisation par fichier texte

La figure 5.5 est un extrait d'un fichier de configuration d'un modèle de comportement : $P(B^k|B^{k-1}S^k_{pres}S^k_{dist}\pi_m)$. Les lignes commençant par le caractère '#' sont considérées des commentaires.

Les autres lignes définissent les enregistrements dans la table. À gauche du signe '=', on retrouve "l'adresse" de l'enregistrement, et à droite entre crochets, sa valeur. L'adresse est définie par la combinaison des valeurs des variables connues $(B^{k-1}, S^k_{pres}S^k_{dist})$. La valeur de l'enregistrement est la distribution de probabilités sur la variable recherchée (B^k).

```
...
# ************************************************************
# Bt-1  Pres    Dist    =      P()
# ************************************************************

# ------------------------------------------------------------
# Bt-1 = Escape
# ------------------------------------------------------------
1       0       0-5 = {0.25, 0.24, 0.24, 1e-2, 0.26}
# objet present:
1       1       0-5 = {x, x, x, 0.39, 0.01}
...
# ------------------------------------------------------------
# Bt-1 = Chase
# ------------------------------------------------------------
3       0       0-5 = {0.24, 0.24, 0.26, 1e-2, 0.25}
# objet present:
3       1       0-3 = {x, x, 0.24, 0.35, 0.01}
3       1       4,5 = {x, x, 0.30, 0.29, 0.01}
...
```

Tab. 5.5 : Paramétrage des modèles en utilisant un fichier texte

Plusieurs enregistrements peuvent avoir la même distribution de probabilités. Par exemple, si le prédateur était en train de fuir ($B^{k-1} = Escape$) et que le prédateur est toujours présent ($S^k_{pres} = 1$) le robot doit continuer de fuir *quelque soit* la distance. Dans un souci de facilité d'utilisation et de lisibilité, une ligne peut définir une liste d'enregistrements. La valeur des variables connues peut être donnée sous forme de liste (le séparateur est la ‘,’), ou sous forme de fourchette (le séparateur est le ‘-’). La distribution de probabilités est alors dupliquée pour chaque enregistrement.

Les distributions doivent être normalisées : la somme des probabilités doit être égale à 1. Ceci peut être contraignant, car lorsqu'un paramètre est modifié, il faut rééquilibrer les paramètres pour conserver la normalisation. Pour cela nous avons introduit l'opérateur ‘x’ comme valeur possible d'une probabilité. Tous les ‘x’ d'une distribution seront remplacés par une valeur identique de façon à ce que la somme soit égale à ‘1’. En prenant comme exemple la deuxième ligne non commentée dans l'extrait, les trois ‘x’ seront remplacés par la valeur 0.2.

5.6.5.2 Initialisation par fichier XML

Certains modèles nécessitent de pouvoir mixer le type de distribution au sein d'une même table. Par exemple le modèle dynamique du filtre maître (cf. tableau 3.2), utilise des listes de probabilités pour $B^{k-1} = O(Obey)$, et une forme “bell-shaped” pour les autres comportements.

Devant ce besoin, nous avons décidé de développer un fichier de configuration plus flexible et plus proche de l'implémentation de programme bayésien avec $ProBt^{©}$. Nous avons choisi comme support le format XML pour répondre au critère de flexibilité. La figure 5.6 est un extrait d'un fichier de configuration XML, d'un modèle dynamique : $P(S^k_{dist}|B^{k-1}S^{k-1}_{dist})$.

Le noeud racine est nommé *plObjects*. Il peut contenir la définition de plusieurs termes. Chaque terme est délimité par un bloc *plComputableObject* et est identifié

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<plObjects>
...
<!--****************************************-->
<!-- P(Sdist | Bt-1 Sdist-1)  -->
<!--****************************************-->
<plComputableObject type="plKernelTable" id="Sdist" desc="P(Sdist | Bt-1 Sdist-1)">
  <Item>
   <plKwn val="4"> <plKwn val="0">
     <plComputableObject type="plProbTable">
       <plProbValues> x, 1e-1, 1e-2, 1e-3, 1e-4, 1e-5 </plProbValues>
     </plComputableObject>
   </plKwn></plKwn>
  </Item>
  <Item>
    <plKwn val="4"> <plKwn val="1">
      <plComputableObject type="plProbTable">
     <plProbValues> x, 0.3, 1e-1, 1e-2, 1e-3, 1e-4 </plProbValues>
   </plComputableObject>
</plKwn></plKwn>
  </Item>
...
  <Item>
    <plKwn val="1-3,5"> <plKwn val="0-5">
      <plComputableObject type="plBellShape">
        <Params mu="[1]" sigma="2"></Params>
     </plComputableObject>
   </plKwn></plKwn>
  </Item>
</plComputableObject>
</plObjects>
```

Tab. 5.6 : Paramètres des modèles : fichier XML

par l'attribut *ID*. Cet attribut est utilisé par le programmeur pour charger le terme souhaité. Le modèle dynamique, donné en exemple, est une table de probabilités (*type="plKernelTable"*). D'où la définition suivante :

<plComputableObject type="plKernelTable" id="S_{dist}" desc="$P(S_{dist} \mid B^{k-1}S - dist^{k-1})$">

Une table de probabilités contient, comme nous l'avons vu, des enregistrements. Ils sont délimités par la balise *Item*. L'"adresse" de l'enregistrement dans la table est toujours définie par les valeurs des variables connues, ici en l'occurrence B^{k-1} et $S - dist^{k-1}$. Une variable connue est définie dans un objet *plKwn* dont l'attribut *Val* est sa valeur. L'adresse de l'enregistrement est ainsi une imbrication d'objets *plKwn* qui se termine avec la définition de la distribution de probabilités.

Comme illustré par l'extrait, la distribution de probabilités est maintenant un objet *plComputableObject* de type variable : *plProbTable*, pour les listes de probabilités, ou *plBellShape* pour une distribution "bell-shaped". L'objet *plBellShape* contient deux paramètres : *Mu* et *Sigma*. La valeur de *Mu* est une redirection ($Mu = [1]$). *Mu* prend la valeur de la variable connue (objet *plKwn*) d'indice 1 (la première variable à l'indice 0).

5.7 Capture et traitement d'image : *bbcam* et *camCGI*

Le rôle du système de capture et de traitement vidéo est de fournir des observations pertinentes permettant au contrôleur de comportements de déduire la présence, la direction et la distance du prédateur, du maître et de la proie.

5.7.1 Spécifications

Comme nous l'avons présenté au chapitre 3, le prédateur, le maître et la proie sont associés à des couleurs : le prédateur est rouge, la proie verte et le maître bleu. Des humains joueront ces rôles en portant des maillots colorés.

Le système de capture et de traitement vidéo doit donc détecter la présence de zone contenant des couleurs prédéterminées. Il doit également évaluer la direction et taille des zones détectées. La taille sera utilisée pour estimer approximativement la distance du sujet.

Dans un souci d'avoir un protocole de communication homogène pour toute l'application, ces observations doivent être disponibles par le biais de requêtes HTTP/CGI.

5.7.2 Schéma d'architecture

Comme illustré par la figure 5.11, le système de capture et de traitement vidéo est composé de trois modules, dont deux que nous avons développés : *bbcam* et *camCGI*.

Blinky, le troisième module, est un outil d'acquisition d'image à partir d'une caméra connectée sur un port IEEE1394 (firewire). Il a été développé par l'équipe MOVI[5] de l'INRIA Rhône-Alpes.

bbcam implémente l'algorithme de détection des centres d'intérêts (cf. 5.7.4.1). Après avoir obtenu une image de *Blinky*, il tente de détecter la présence, la direction et la taille des zones, et les sauvegarde dans une mémoire partagée. Ce module contrôle également les mouvements de la caméra par le biais du système pan-tilt. La position de la caméra est aussi sauvegardée en mémoire partagée.

Comme son nom l'indique, *camCGI* est un CGI[6] géré par un serveur web, en l'occurrence *Apache*©. Il récupère les données stockées dans la mémoire partagée pour les encapsuler dans une page HTML. Cette page est renvoyée au client qui a généré l'appel au CGI.

5.7.3 Exécution du système

Pour démarrer le système d'acquisition et de traitement vidéo, les modules doivent être lancés dans l'ordre suivant : *Blinky*, puis *BBCam*. L'exécution de *camCGI* est quand à elle gérée par le serveur web. *Blinky* est le premier à être démarreé car il est en charge des acquisitions d'images pour servir *BBCam*.

Blinky

La ligne de commande pour démarrer *Blinky* est :

[5] http ://www.inrialpes.fr/movi

[6] Common Gateway Interface (http ://fr.wikipedia.org/wiki/CGI)

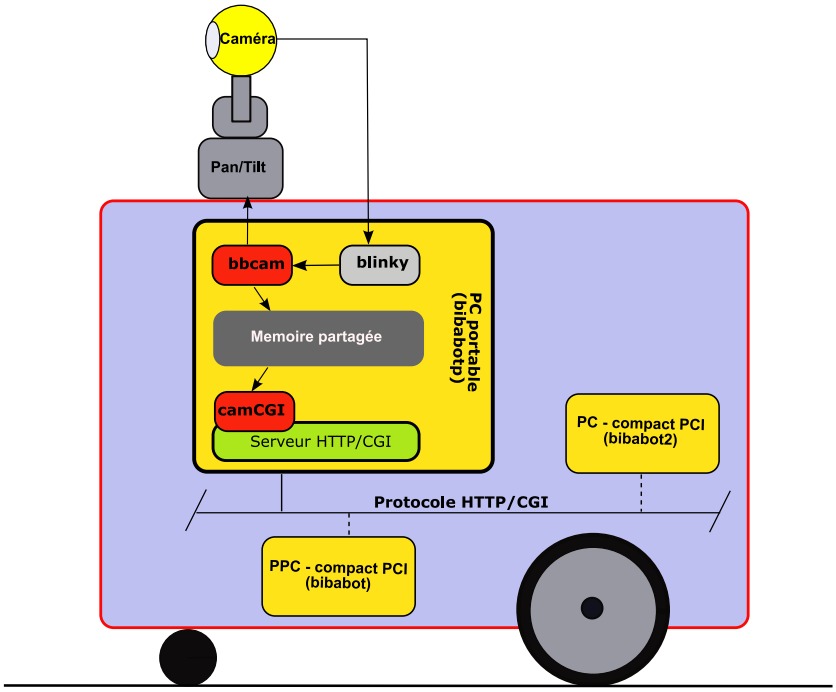


Fig. 5.11 : Schéma d'architecture du système d'acquisition et de traitement d'image

```
$ blinkyf1394 -p<port no> -c<caméra N°> -d<device no>&
```

Les paramètres de la ligne de commande de *Blinky* sont recensés dans le tableau 5.7. *Blinky* peut générer une liste des caméras connectées avec leurs identifiants <*port no*> et <*caméra N°*>, en exécutant la commande ***blinkyf1394***.

Paramètres	Description
<port N°>	Identifiant du port IEEE1394 sur lequel est branché la caméra
<caméra N°>	Identifiant de la caméra sur le port
<device N°>	*Blinky* est capable de gérer plusieurs caméras. Les clients doivent se connecter à <*device N°*> pour obtenir les services de *Blinky* concernant la <*caméra N°*> sur <*port N°*>

Tab. 5.7 : Paramètres de la ligne de commande de *Blinky*.

BBCam

La ligne de commande pour démarrer *BBCam* est donnée ci-dessous. Le paramètre <*device no*> doit être le même que celui de *Blinky*.

```
$ bbcam <device no>
```

Nous allons maintenant présenter l'implémentation des modules que nous avons développés : *BBCam* et *camCGI*.

5.7.4 Le module *BBCam*

Le module *BBCam* est construit sous forme d'un daemon (ou service) Linux. Après une phase d'initialisation, il implémente une boucle de traitement dans laquelle une image est analysée, la caméra est déplacée et la mémoire partagée mise à jour. L'algorithme ci-dessous donne l'enchaînement des tâches.

Algorithme 1 : algorithme du module BBCam

```
1  Lecture du fichier de configuration;
2  Initialisation de la réception de signaux;
3  Initialisation du processus de détection;
4  Initialisation de la mémoire partagée;
5  Initialisation du système pan-tilt;
6  tant que non réception d'un signal de fin faire
7  |   Mise à jour de la position du système pan-tilt en mémoire partagée;
8  |   Acquisition et détection des zones de couleur;
9  |   Mise à jour des informations des zones en mémoire partagée;
10 |   Nouvelle position pan-tilt;
11 fin
```

L'initialisation de la réception de signaux consiste à masquer ou à valider la réception des signaux système. Par exemple, le signal SIGTERM, généré par la commande *kill*, est validé pour mettre fin à la boucle de traitement et terminer proprement l'exécution de *BBCam*.

Nous présentons dans les sections qui suivent, les algorithmes de détection des zones et de déplacement de la caméra, suivis de l'analyse du fichier de configuration du système d'acquisistion et de traitement d'image.

5.7.4.1 Détection des zones

La tâche d'*acquisition et de détection des zones* est implémenter dans une librairie nommée *visionLib*. Elle a été développée par Soraya Arias membre de l'équipe SED[7] de l'INRIA Rhônes-Alpes. Cette librairie implémente également une interface permettant de visualiser en temps réel les images traitées avec, en surimpression, les zones détectées. Cette interface est illustrée par la figure 5.12.

Cet algorithme utilise la librairie *openCV* (version v0.9.6). La détection consiste à délimiter une zone contenant une couleur préalablement spécifiée (cf. fichier de configuration, section 5.7.4.3). Une couleur est définie par un histogramme 2D construit à partir des intervalles de teinte et de saturation qui la caractérise.

L'algorithme exécute séquentiellement les actions suivantes :

1. Acquisition d'une image. Deux résolutions sont possibles : 640x480 ou 320x240.
2. Back-projection de l'image. Cette étape consiste à comparer l'histogramme de chaque pixel de l'image avec celui de couleur recherchée. On obtient une image en noir et blanc ou les pixels blancs signifient que la teinte et la saturation du pixel étaient comparables à celles de la couleur recherchée.

[7]Supports expérimentaux et développement logiciel.

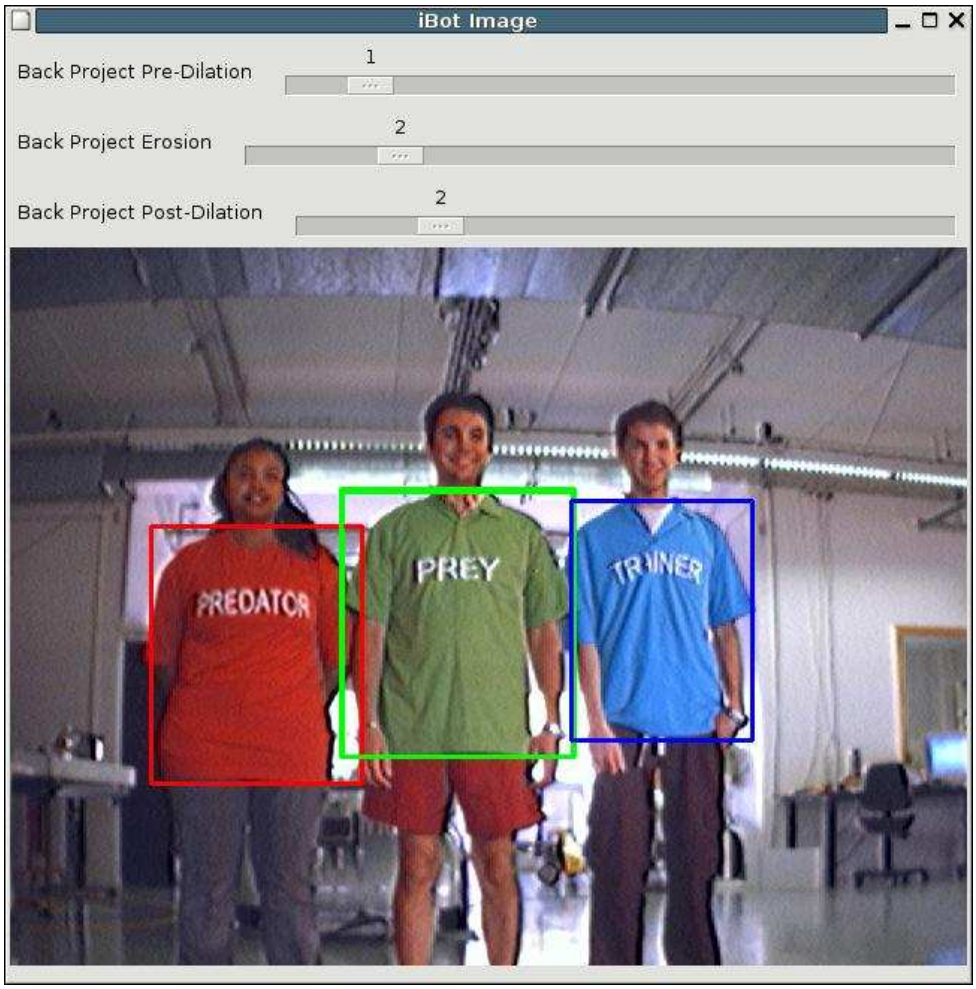


Fig. 5.12 : Détection des trois teeshirts

3. Filtrage de l'image noir et blanc pour supprimer les pixels parasites.
4. Recherche des contours des zones blanches restantes.
5. Sélection de la zone dont la surface est la plus grande. Cependant, pour qu'une zone soit considérée valide, elle doit avoir une surface minimale, 1/100ème de la taille de l'image, et un ratio hauteur/largeur supérieur à 0.5.
6. Calcul du contour rectangulaire englobant la zone sélectionnée.
7. Évaluation de la position de la zone dans l'image, en coordonnées angulaires (radians). L'origine (0°,0°) est le centre de l'image. Le point de référence de la zone est son centre de gravité.

5.7.4.2 Contrôle du système pan-tilt

Il a été fait le choix d'orienter la caméra toujours vers la zone dominante, celle ayant la surface la plus grande. L'algorithme de contrôle du système pan-tilt suit les étapes suivantes :

1. Sélection, si elle existe, de la zone dominante.

2. Calcul de la position finale de système pan-tilt par rapport au coordonnées de la zone sélectionnée.
3. Déplacement du système pan-tilt vers la position calculée. Afin de rendre plus stable les mouvements de la caméra, le déplacement est effectué seulement s'il est supérieur a 3°, en pan ou en tilt.

5.7.4.3 Le fichier de configuration

Le système d'acquisition et de traitement d'images est configuré par le fichier *config.xml*. La figure 5.8 en propose un extrait. Il est composé de trois sections principales : *camera*, *color* et *area*.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <config>
    <camera name="IBot">
      <videomode>
        <mode>640x480_RGB</mode>
        <framerate>15</framerate>
      </videomode>
      <property name="WHITE_BALANCE">
        <one_push>0</one_push>
        <auto_func>1</auto_func>
        <on_off>1</on_off>
        <value_a>0</value_a>
        <value_b>0</value_b>
      </property>
      <property name="SHARPNESS" on_off="1"></property>
      <image>
        <width>640</width>
        <height>480</height>
      </image>
      <angleview unit="degre">
        <x>26.0</x>
        <y>21.0</y>
      </angleview>
    </camera>
    <color type="red">
      <range><hue>110,120</hue><sat>100,255</sat></range>
      <range><hue>5,20</hue><sat>130,240</sat></range>
    </color>
    <color type="green">
      <range><hue>110,120</hue><sat>100,255</sat></range>
      <range><hue>5,20</hue><sat>130,240</sat></range>
    </color>
    <color type="blue">
      <range><hue>110,120</hue><sat>100,255</sat></range>
      <range><hue>5,20</hue><sat>130,240</sat></range>
    </color>
    <area>
      <minsize>1.0/200.0</minsize>
      <ratio>1.0/2.0</ratio>
    </area>
  </config>
```

Tab. 5.8 : Fichier de configuration du système d'acquisition et de traitement vidéo

La section *camera* définit les paramètres d'initialisation de la caméra. Les sous-objets *videomode* et *property* sont des paramètres pour l'acquisition d'une image, et sont donc envoyés à *Blinky*. Les sous-objets *image* et *angleview* sont utilisés par le

programme d'analyse d'image, entre autres pour pouvoir évaluer la direction de la zone connaissant les angles de vue horizontale et verticale de la caméra.

Les objets *color* permettent de définir les intervalles de teinte et de saturation d'une couleur. Ces informations sont utilisées pour la détection des zones (cf. section 5.7.4.1).

La section *aréa* permet de définir la surface et le ratio hauteur/largeur minimal pour qu'une zone soit considérée valide.

5.7.5 Le module *CamCGI*

CamCGI est un CGI dont l'exécution est gérée par un serveur web. Son rôle est de récupérer les données stockées dans la mémoire partagée pour les encapsuler dans une page HTML. Cette page est renvoyée au client qui a généré l'appel au CGI.

Le format de la page est donné en exemple dans la figure 5.9. Elle contient :

- la description des zones détectées uniquement, avec l'identifiant de la couleur associée, la direction horizontale et verticale, ainsi que la taille relative à l'image,
- la position du système pan-tilt.

Chacune de ces informations est estampillée pour contrôler leur validité.

```
<HTML>
 <BODY>
  <HR>
   <P>
    <PRE>
     <Couleur1>, <TS_sec>, <TS_usec>, <thethaX>, <thetaY>, <size>
     <Couleur2>, <TS_sec>, <TS_usec>, <thethaX>, <thetaY>, <size>
     ...
     <TS_sec>, <TS_usec>, <panpos>, <tiltpos>
    </PRE>
   </P>
  </HR>
 </BODY>
</HTML>
```

Tab. 5.9 : Format de la page HTML

Chapitre 6

Résultats et validation

Ce chapitre présente les résultats de l'exécution du programme décrit au chapitre 3. Ces résultats sont comparés au comportement défini dans le cahier des charges, et quelques remarques sur la mise en oeuvre du canevas proposé sont présentées.

6.1 Présentation des résultats

Le développement et l'évaluation de l'application se sont déroulés de façon incrémentale, en combinant un cycle de vie globale de type spirale, proposée par B. Boehm, et pour chaque pas de la spirale un cycle de vie en **V** pour mettre en évidence les tests unitaires, les tests d'intégration et de qualification.

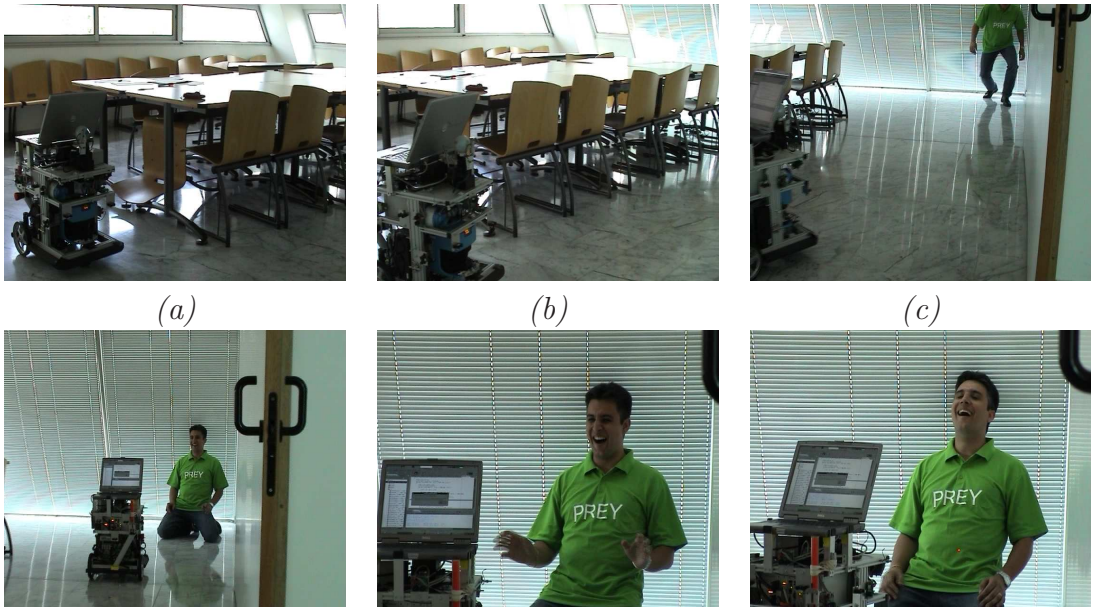
Initialement, chaque comportement a été testé séparément : chasser la proie, s'échapper du prédateur, rester immobile pour ne pas être vu, suivre le maître et glaner. Pour cela, les valeurs des variables de comportement ont été fixées, ce qui a eu pour conséquence d'inhiber le modèle de comportement. Ce travail a permis d'ajuster, une à une, les actions motrices associées à chaque comportement, et ainsi décomplexifier le déverminage.

La deuxième étape a été de tester la combinaison de comportements. Tout d'abord l'évitement d'obstacles à été validé avec chaque comportement. Cette phase a permis de certifier que le module d'évitement d'obstacles était toujours prioritaire sur les actions motrices, peu importe le comportement désiré. Une fois la partie sécurité assurée, l'association de comportement a été testé, dans le but d'affiner les réglages des modèles de comportements, ainsi que la fusion des propositions de comportements et d'actions motrices.

Nous présentons maintenant les résultats de deux comportements testés unitairement, dont un avec évitement d'obstacles, suivi de deux expérimentations de combinaison de comportements. Ces expériences ont été réalisées dans différentes pièces de l'INRIA avec des conditions d'éclairage hétérogène.

Chasse et capture

Chasser et capturer une proie (*chase*) fait partie du comportement spécifié. Les résultats de ce comportement sont illustrés par la séquence de photos 6.1. La première photo montre le robot glanant, puis il détecte une proie sur sa gauche. Cette photo



(a) *(b)* *(c)*

(d) *(e)* *(f)*

Fig. 6.1 : Comportement : chasse. Description des photos : (a) Le robot se déplace librement en évitant les obstacles ; (b) Il détecte une proie sur son côté gauche ; (c) Le robot se déplace en direction de la proie ; (d) Il approche la proie autant que possible ; (e) Le robot s'arrête pour capturer la proie ; (f) La proie est capturée avec le pointeur laser.

et la suivante montrent que même lorsque le corps du robot change d'orientation pour éviter des obstacles (les tables), la caméra reste focalisée sur la proie.

Le robot accélère lorsque la proie est éloignée et pour des raisons de sécurité, il diminue progressivement sa vitesse à mesure qu'il s'en rapproche. Lorsqu'il se trouve à proximité de la proie (avant dernière photo) le robot s'arrête et la capture en actionnant le pointeur laser (dernière photo).

Obéissance avec évitement d'obstacles

Le comportement *obey* signifie suivre le maître. La séquence de photos 6.2 illustre comment le robot exécute ce comportement. Dès l'instant où le robot aperçoit le maître, il le suit et tente de s'en rapprocher (photos du haut). Les trois photos du bas montrent le robot évitant un obstacle (la table basse) qui est sur sa trajectoire tout en conservant la caméra fixée sur le maître.

Réactivité face au danger

Le robot doit être réactif quand il se trouve face à un prédateur. Il peut rester immobile si le prédateur est suffisamment éloigné, ou alors s'échapper si le prédateur est trop près. Il peut encore chercher la protection du maître si celui-ci est présent.

La séquence de photos 6.3, montre le robot poursuivant sa proie lorsqu'un prédateur arrive (troisième photo). Son ennemi est très proche, il fait immédiatement

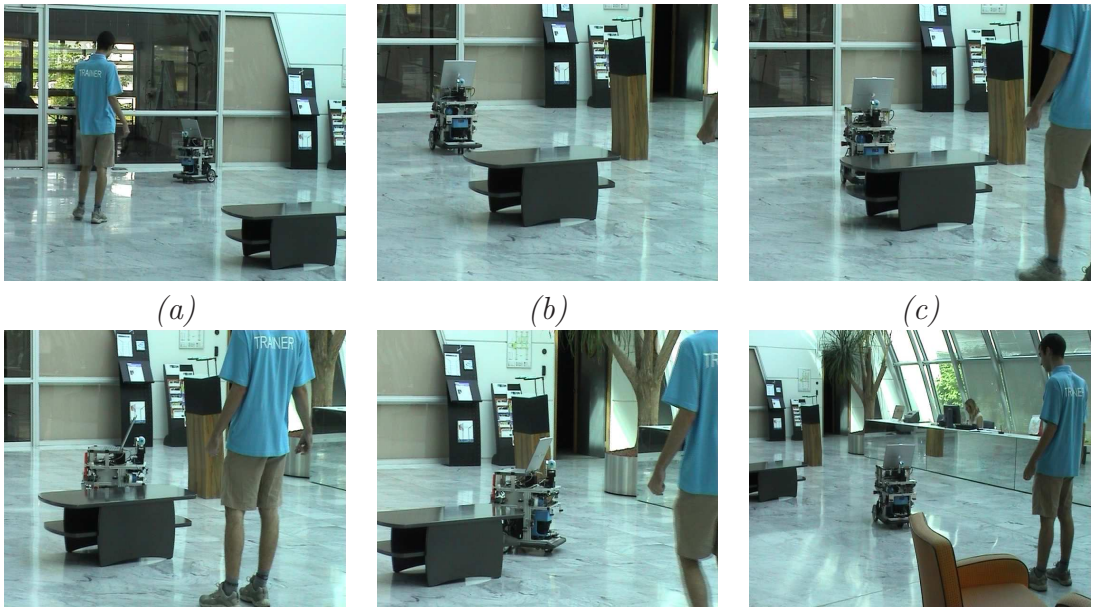


Fig. 6.2 : Comportement : obéissance avec évitement d'obstacles. Description des photos : (a) Le robot détecte le maître ; (b) Il suit son maître ; (c) Un obstacle est sur la trajectoire ; (d) Le robot change de direction pour éviter l'obstacle ; (e) Il contourne l'obstacle tout en « regardant » son maître ; (f) Le robot s'approche de son maître.

demi-tour pour tenter de s'échapper. En cherchant à suivre la route de fuite, il se retrouve à l'abri dans un coin, caché du prédateur.

Coordination de comportements

Cette expérience met en oeuvre deux comportements opposés : s'échapper du prédateur et suivre le maître. La situation présentée par la séquence de photos 6.4, commence avec le robot glanant (*wandering*) lorsqu'apparaît sur sa gauche, assez éloigné, un prédateur. Le robot s'arrête et reste immobile pour espérer ne pas être découvert (troisième photo).

Il reste immobile jusqu'à ce que le maître entre en scène. Le robot se sent en sécurité, alors il suit le maître même si ce dernier l'emmène à côté du prédateur.

6.2 Analyse des résultats

Comparaison du comportement implémenté avec celui spécifié

Le comportement implémenté sur le robot a été comparé au comportement spécifié dans le cahier des charges. Les résultats sont considérés satisfaisants. L'exécution des comportements séparés, sans et avec évitement d'obstacles, et les combinaisons de comportements complexes sont compatibles avec le comportement spécifié.

Cependant, il est à noter la vitesse relativement lente du robot. Le robot doit interagir avec des humains, et une vitesse limitée est plus sûre. Cependant, la vitesse

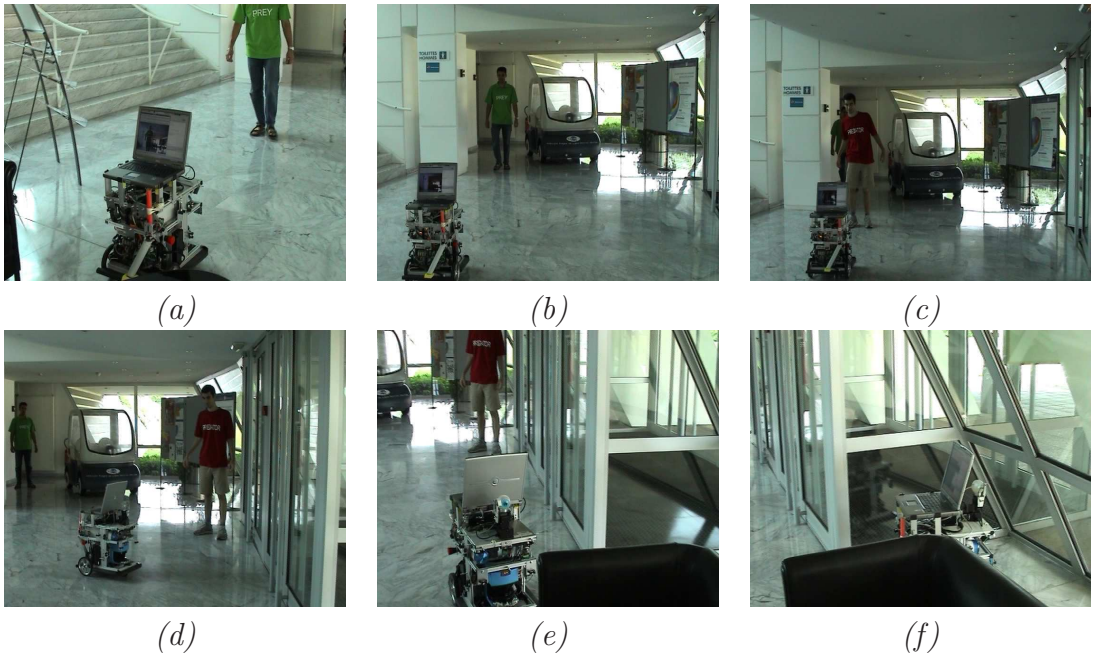


Fig. 6.3 : Comportement : chasse puis réactivité face au prédateur.Description des photos : (a) Le robot détecte une proie ; (b) Il poursuit la proie ; (c) Un prédateur apparaît ; (d) Le robot s'arrête de chasser et cherche une route de fuite ; (e) Le robot s'échappe ; (f) Il se cache du prédateur.

du robot a également été bridée à cause du temps de cycle du programme : un nouvel ordre moteur est obtenu chaque 250 millisecondes. Afin de permettre des vitesses de mouvement plus élevées, le robot doit être plus réactif à son environnement. Un temps de cycle de 100 millisecondes est considéré raisonnable pour un robot partageant son environnement avec des humains.

Les processus les plus consommateurs de ressources CPU, sont les calculs probabilistes et le traitement vidéo. Les calculs probabilistes sans approximation prennent au total 90 millisecondes. De plus, une partie de ces calculs est faite en parallèle avec l'acquisition des données sensorielles, donc en temps masqué. Le goulot d'étranglement est le processus de vision : le temps de cycle est 200 millisecondes.

La caméra est montée sur un système pan-tilt, comme illustré sur la photo 2.1. Un cycle complet de vision implique la capture et le traitement d'une image, et l'orientation du système pan-tilt pour centrer la caméra sur le sujet le plus proche. Après avoir déplacé la caméra, il est nécessaire d'attendre 50 millisecondes pour que la prochaine image capturée soit suffisamment nette.

Dans l'effort de réduire le temps cycle de la vision, quelques pistes sont explorées. Une première idée est de tenter de réduire le temps de stabilisation de l'image en utilisant une caméra plus performante, mais les réglages sont plus complexes. Une deuxième piste est d'optimiser le traitement d'une image et intégrer la notion de « tracker » et ainsi réduire la taille de la zone de détection sur une image. Ces pistes présentent de bonnes perspectives, malheureusement elles n'avaient pas encore abouti à la fin du stage.

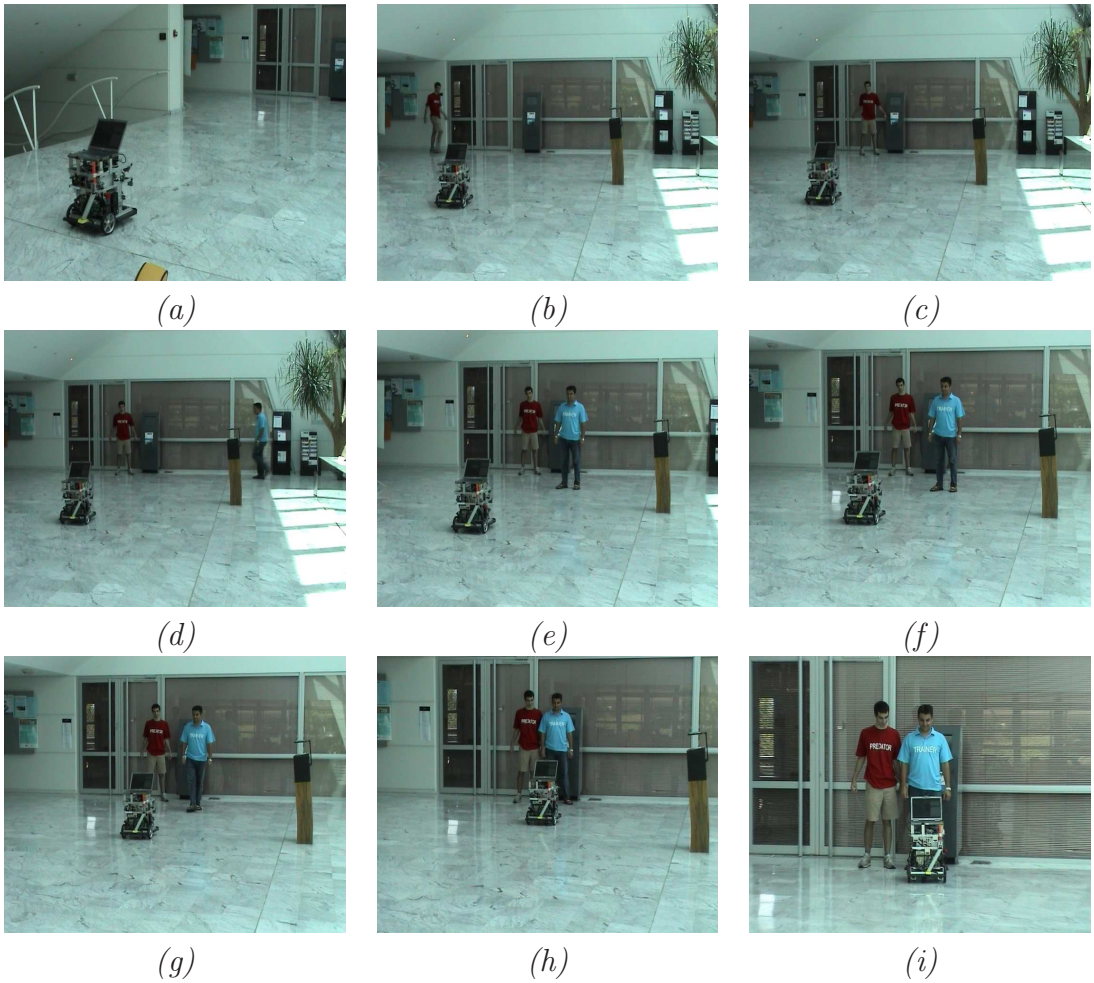


Fig. 6.4 : Comportement : coordination. Description des photos : (a), (b) Le robot se déplace librement ; (c) Il détecte assez loin un prédateur ; (c), (d) Il reste immobile ; (e) Il détecte son maître qui approche ; (f), (g), (h) Le robot suit son maître ; (i) Même en présence du prédateur, le robot reste à côté de son maître.

Méthode de programmation

L'implémentation du comportement spécifié sur un robot réel en utilisant le canevas proposé, n'est pas particulièrement difficile. Le canevas impose une méthode de programmation modulaire et incrémentale.

En dépit du grand nombre de variables et de la taille de la distribution conjointe, chaque filtre est un module indépendant et autonome qui peut être développé et validé séparément. Un filtre est lui même composé de plusieurs modèles : dynamique, capteur, comportement et moteur. L'indépendance de ces modèles est suffisante pour permettre le choix des formes paramétriques, l'identification des paramètres et la validation de façon unitaire.

Toutefois, la tâche peut devenir un peu plus complexe lors de l'association des filtres, et plus précisément de la fusion des variables communes. Ces variables sont définies par une distribution de probabilités globale issue de la fusion des propositions

de chaque les filtre. Il peut alors être nécessaire d'équilibrer les valeurs de probabilité dans les filtres, de façon à obtenir au final le comportement escompté. Par contre, si plus qu'un « affinage » est nécessaire, alors il y a très probablement une insuffisance dans la description des modèles.

Les paramètres des modèles et des tables de probabilités doivent être choisis, autant que possible, uniquement avec les connaissances locales du filtre. Chasser ou pas, ne dépend que des connaissances locales du filtre proie. Le filtre est, dans ces cas, certain de ses propositions : les comportements recommandés ont des valeurs de probabilités fortes, et des valeurs faibles pour les comportements interdits.

Cependant, dans certains cas, les connaissances locales ne sont pas suffisantes pour interdire ou recommander un comportement. Le filtre prédateur ne peut interdire de suivre le maître, car il peut être recommandé par le filtre maître comme une bonne alternative en présence du prédateur. De la même façon, le filtre maître ne peut recommander la chasse, car la proie n'est peut être pas présente. Dans ces deux exemples, la présence du maître et de la proie ne font pas partie des connaissances locales, respectivement des filtres prédateur et maître. Pour ce type de situation, les filtres ne doivent pas donner d'avis ; ceci se traduit par des valeurs de probabilités moyennes.

Utilisation pratique du canevas

L'utilisation de ce modèle de programmation de comportements sur un robot réel, suivant la méthode présentée dans la section précédente, a permis d'obtenir un comportement proche de celui spécifié.

Le robot évolue de façon autonome dans un environnent partagé avec des humains. Pour sa sécurité et celle de son environnement, un module d'évitement d'obstacles a été couplé au contrôleur de comportements. Autour de ce noyau il a bien entendu été nécessaire de développer une interface pour le traitement des données sensorielles, dont la vision, et des commandes motrices.

Le système ainsi obtenu a permis d'obtenir des résultats très encourageants pour continuer a faire mûrir et à étendre le canevas proposé par [Koi05].

Chapitre 7

Conclusion et perspectives

L'objectif de nos travaux était l'implémentation d'un comportement préalablement spécifié, celui d'un animal, sur le robot BIBA. Nous devions pour cela utiliser le canevas proposé par [Koi05] et valider qu'il pouvait être utilisé en pratique sur un robot.

Le robot devait pouvoir se déplacer librement dans un environnement fermé, tout en évitant les obstacles. Il interagit avec quatre centres d'intérêts : un prédateur, une proie, son maître et une route de fuite. Face à un environnement inconnu et imprévisible, le robot devait être capable de traiter l'incomplétude inhérente du modèle de l'environnement ainsi que l'incertitude.

7.1 Bilan

Nous avons dans un premier temps défini la programmation bayésienne comme une méthodologie et un formalisme pour le développement de logiciels qui soient capables de traiter des informations incomplètes et incertaines.

Nous avons ensuite exposé la méthode proposée par [Koi05] pour le contrôle d'un système sensori-moteur. Cette approche propose d'associer à chaque centre d'intérêt, évoluant dans l'espace des mouvements du robot, une extension du filtre bayésien : le *filtre élémentaire*. Il est élémentaire non pas pour traduire une quelconque simplicité dans sa sémantique, mais pour rappeler que chacun de ces filtres est un élément du programme bayésien global.
À un instant t, chaque filtre élémentaire doit estimer la position de son centre d'intérêt, en déduire un comportement approprié et finalement proposer les commandes motrices adéquates. Le comportement que le robot doit exécuter est le résultat de la fusion des propositions de chaque filtre.

Puis, nous avons détaillé le contrôleur de comportements composé de quatre filtres sensori-moteurs élémentaires : prédateur, proie, maître et route de fuite. Nous avons également explicité le choix des formes paramétriques et de leurs paramètres pour les modèles dynamique, capteur, comportement et moteur.

Le rôle du module d'évitement d'obstacles a également été détaillé : transmettre au système d'asservissement de bas niveau, autant que possible, les commandes choisies par le contrôleur de comportements, tout en garantissant la sécurité du robot. Nous avons pu apprécier la simplicité et la clarté de la décomposition en sous-

problèmes simples (des zones) dont les résultats sont ensuite fusionnés.

Finalement, après avoir détaillé le développement de ces modules, nous avons analysé les résultats obtenus. L'utilisation de ce modèle de programmation de comportements sur un robot réel, a permis d'obtenir un comportement proche de celui spécifié. De plus, le canevas proposé impose une méthode de programmation modulaire et incrémentale. En dépit du grand nombre de variables et de la taille de la distribution conjointe du programme global, chaque filtre est un module indépendant et autonome qui peut être développé et validé séparément.

Ce sont des résultats très encourageants pour continuer à faire mûrir et à étendre le canevas proposé par [Koi05].

7.2 Les perspectives

Au cours du développement de cette application, beaucoup d'idées d'amélioration ou d'évolution sont apparues.
Nous en avons déjà cité certaines en deuxième partie du chapitre 6.

La performance de la perception visuelle

Il serait intéressant de continuer à améliorer la qualité et le temps de cycle de la perception visuelle. En effet, la caméra est la seule source d'informations sensorielles pour la plupart des filtres élémentaires. Pour avoir un comportement robuste, les observations visuelles doivent être aussi fiables que possible.

Le premier pas essentiel est d'investir sur une caméra haut de gamme. Des tests révélateurs ont été réalisés avec une AVT MARLIN : les images étaient plus nettes et la balance des blancs plus performante.

Pour diminuer sensiblement le temps de cycle, des changements plus drastiques doivent être apportés à l'algorithme. Une piste serait d'implémenter la notion de *tracker*. Elle permettrait de ne traiter qu'une partie de l'image : une zone légèrement plus grande que celle occupée par le sujet sur l'image précédente.

Une deuxième piste serait d'implémenter le mécanisme de sélection d'attention proposé également par [Koi05]. Il permet d'optimiser l'utilisation de capteurs "lourds", tels que la caméra et le télémètre laser, en focalisant l'attention du robot sur un sujet. Par exemple, lorsque le robot fuit il n'est pas nécessaire qu'il prête attention à la présence de la proie. On économise ainsi un traitement d'image sur un temps de cycle.

Un outil de « monitoring »

Le robot traite une très grande quantité d'informations pour sélectionner un comportement. Il est parfois difficile d'expliquer le choix d'un comportement, visuellement il paraît stupide. Il serait très utile au programmeur de pouvoir visualiser, en temps réel, la valeur de certaines variables ou la courbe de distribution de probabi-

lités préalablement sélectionnée. Cet outil serait une extension des fichiers de traces et des courbes *Gnuplot*© présentés au chapitre 5.

Cet outil pourrait également enregistrer une séquence de comportements exécutés par le robot avec les paramètres associés (observations, états, etc), afin d'être rejouée ultérieurement dans le but de déverminer le programme.

Le fichier d'initialisation

Les fichiers d'initialisation des modèles du filtre élémentaire se sont avérés très pratique pour le développement du contrôleur de comportements. En effet, lors du changement d'un paramètre il n'est plus nécessaire de recompiler le code source et de redéployer les codes binaires.

Il serait intéressant d'étendre les possibilités du fichier d'initialisation XML, et surtout de la classe utilitaire associée, pour pouvoir définir n'importe quel type de forme paramétrique proposée par l'API de *ProBT*©. Plus encore, pouvoir définir la décomposition d'un modèle, voire même pouvoir décrire les questions d'utilisations, en claire pouvoir implémenter un programme bayésien complet dans ce fichier XML, apporterait des avantages conséquents.

Il ne serait plus indispensable d'être un programmeur chevronné en C++ pour pouvoir implémenter un programme bayésien. Moyennant une interface graphique bien "pensée" capable de générer le fichier, il ne serait même pas nécessaire d'avoir des connaissances XML. L'utilisateur pourrait se concentrer totalement sur son modèle bayésien, sans se préoccuper de l'allocation de pointeurs ou de balisages XML.

Conclusion

Le comportement relativement simple que nous avons implémenté sur le robot BIBA a permis d'entrevoir les possibilités du canevas proposé par [Koi05]. Il serait intéressant de continuer à ajouter des nouveaux comportements, comme par exemple le retour à la base. Ce comportement peut être implémenté en suivant l'approche proposée dans [MAK05], où le robot dépose des jetons pendant ses déplacements afin de retrouver son chemin de retour.

Une deuxième approche serait de connecter le robot, via le réseau sans fils, à la plate-forme de vidéo-surveillance détaillée dans [Bon06]. Elle possède un serveur de carte, qui tient à disposition le plan détaillé d'une partie des locaux de l'INRIA. Par le biais de caméras et de logiciels de suivi de cibles (personnes, véhicules, etc.), le serveur ajoute à la carte la position des cibles détectées. Le robot pourrait télécharger le plan des lieux et ainsi se déplacer tout en connaissant sa position et celle de sa base. La caméra du robot pourrait également servir de caméra ambulante pour la plate-forme de vidéo-surveillance.

Annexe

Annexe A

Guide utilisateur

Ce document à pour objectif de guider autant que possible l'utilisateur à exécuter l'expérience développée tout au long de ce stage. Le comportement implémenté sur le robot BIBA est rappelé brièvement en section A.1.

Mais avant de mettre en route le robot, prenez le temps de réfléchir à ce que vous voulez montrer à vos spectateurs. Voulez vous montrer l'obéissance du robot à son maître ? La chasse d'une proie ? La fuite face à un prédateur ? Ou tout simplement l'évitement d'obstacles ? Mettez en place un scénario le plus précis possible. Choisissez alors la scène de la démonstration et sélectionnez les acteurs. Ils doivent impérativement utiliser l'habillage prévu à cet effet : le polo rouge désigne le prédateur, le vert la proie et le bleu le maître. Cette réflexion préalable vous fera gagner du temps et économisera les batteries du robot.

A.1 Description de l'expérience

Le robot se déplace librement dans un environnement fermé tout en évitant les obstacles. Il interagit avec quatre centres d'intérêts : un prédateur, une proie, son maître et une route de fuite.

Le prédateur, la proie et le maître sont associés à des couleurs et sont détectés para l'analyse du flux vidéo de la caméra : le prédateur est rouge, la proie verte et le maître bleu. Des humains joueront ces rôles en portant des maillots colorés.

Face à un prédateur, il tente de se protéger. Près de son ennemi, il cherche à s'échapper, si une route de fuite est trouvée. Éloigné de celui-ci, il reste immobile pour ne pas être vu.

En présence du maître, le robot se sent en sécurité. Il ne fuit pas les éventuels prédateurs. Le robot suit son maître tant qu'il le perçoit. Si le robot détecte une proie il la prend en chasse, et lorsqu'elle est assez proche, il la « capture ».

A.2 Présentation du robot

A.2.1 Le robot Biba

Le robot Biba, illustré sur la figure A.1, a été développé par *Bluebotics*[1]. Il est de taille moyenne (50 cm) et pèse environ 30 kg. Il est équipé d'une caméra montée sur un système pan-tilt et d'un télémètre laser situé sur l'avant du robot. Il est également équipé de capteurs de proximité infrarouges et à ultrasons. Les forces d'inertie appliquées au robot peuvent être mesurées grâce à un système vestibulaire. Enfin, des entrées-sorties, digitales et analogiques, RS422 et I2C, offrent la possibilité d'étendre le système.

Le robot dispose d'un bloc avec un bus fond de panier « Compact PCI ». Sur ce bus, sont connectés une carte alimentation, deux cartes entrées-sorties et un PowerPC. Parce qu'il ne possède pas de disque dur, le PowerPC télécharge son fichier de démarrage sur un serveur. *Bluebotics*, fournisseur du « boot-file », a fait le choix d'utiliser *XO/2*[2] comme système d'exploitation temps réel.

Par nécessité de puissance de calcul, un deuxième calculateur « Compact PCI » a été ajouté. Contrairement au PowerPC, celui-ci est un pc complet, avec une carte graphique, un disque dur, trois ports USB et deux séries, deux cartes éthernet et une entrée PS/2. Il est doté d'un microprocesseur à fréquence variable (600 Mhz à 1.6 Ghz).
Enfin, un emplacement a été aménagé pour accueillir un PC portable, et ainsi augmenter les ressources du système.
Nous avons fait le choix d'utiliser *Linux* comme système d'exploitation. Ainsi le PC « Compact PCI » et le portable ont été installés avec une distribution Debian Sarge.

A.2.2 Distribution des tâches

Comme illustré par la figure A.2, trois calculateurs se répartissent les trois tâches principales de l'expérience : la gestion de bas niveau des données capteurs et commandes motrices, le traitement vidéo et le contrôle de haut niveau de la sécurité et du comportement du robot. Le support de communication est un protocole HTTP/CGI.

Le robot Biba a été livré par la société *Bluebotics* avec un contrôleur de bas niveau : le PowerPC, nommé *bibabot*. Sur un système d'exploitation temps réel, *XO/2*, le PowerPC implémente une couche d'abstraction de bas niveau. Ainsi les données sensorielles et les commandes motrices sont accessibles via une interface HTTP/CGI. Il intègre également un module de sécurité basique capable de débrayer les moteurs lorsque les “bumpers” du robot entrent en contact avec un obstacle.

Le portable, nommé *Bibabotp*, seul à posséder une entrée firewire, est en charge

[1] www.bluebotics.com
[2] http ://xo2.org

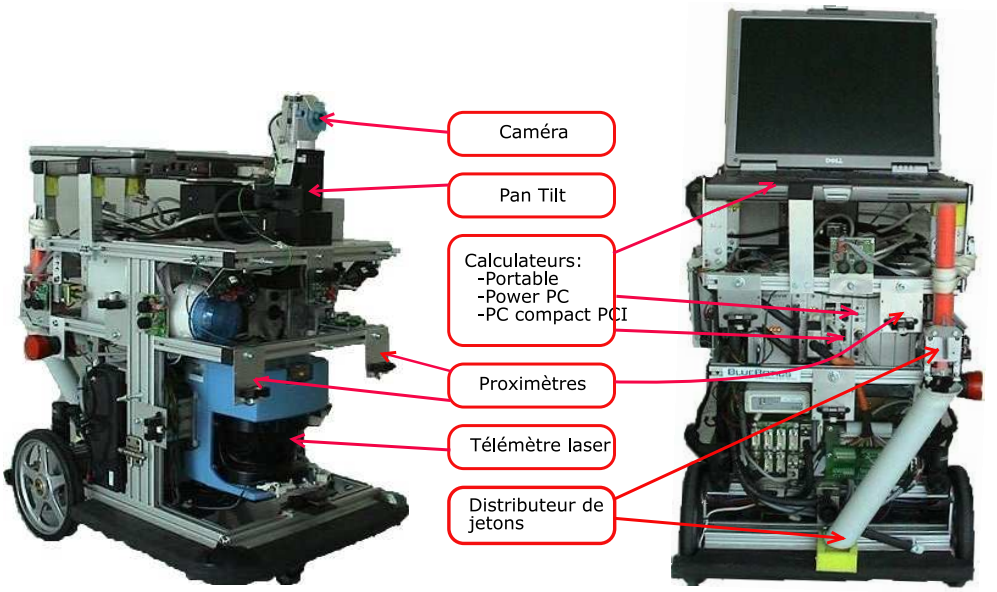


Fig. A.1 : Le robot Biba développé par *Bluebotics*(www.bluebotics.com)

de la capture et du traitement des données massives de la caméra. Ce sera son unique rôle car ces tâches sont gourmandes en ressources CPU et mémoire. Dans un souci d'avoir un protocole de communication homogène pour toute l'application, un serveur web a été installé sur *Bibabotp* (ie Apache). De cette façon, les données sensorielles issues du traitement vidéo sont également disponibles par le biais de requêtes HTTP/CGI. *bibabotp* est également le serveur de fichiers de démarrage pour le PowerPC.

Enfin, *bibabot2* joue le rôle de contrôleur de haut niveau. Il est responsable du comportement général du robot. A chaque pas de temps il évalue le nouveau comportement à adopter en accord avec les données capteurs fournies par les deux précédents calculateurs. Grâce à son module d'évitement d'obstacles, il garantit également que les commandes motrices envoyées au système d'asservissement soient sûres.

A.3 Démarrer une démonstration

Avant d'aller plus loin, vous devez avoir mis en place un scénario, choisi la scène de la démonstration et sélectionné les acteurs. Le prédateur a le polo rouge, la proie a le vert et le maître a le bleu.

Si ces conditions sont remplies, vous êtes prêt pour la première étape : le démarrage du robot.

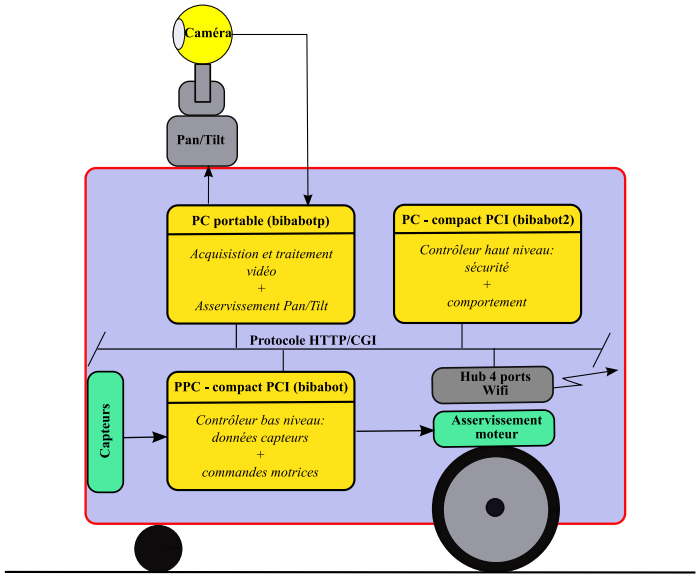


Fig. A.2 : Structure des calculateurs embarqués

A.3.1 Le démarrage du robot

Nous entendons par démarrage, la mise sous tension du robot et l'initialisation du contrôleur de bas niveau : le PowerPC. Rappelons que ce dernier est configuré pour télécharger son fichier de démarrage du PC portable. Nous comprenons donc que l'installation du PC portable sur le robot soit la première chose à faire avant même sa mise sous tension.

A.3.1.1 Installer et démarrer le PC portable

Commencez par poser le PC portable dans l'emplacement aménagé sur le haut du robot. Ensuite, comme illustré par la figure A.3, connectez les éléments suivants :

- la carte pcmcia firewire 3 ports,
- le câble d'alimentation à la carte firewire,
- la caméra à la carte firewire,
- le câble réseau,
- le câble série de la pantilt.

Enfin, démarrez le portable et identifiez-vous en tant que « *demo* » (mot de passe « *demo05* »).

Important : Afin d'économiser la batterie, gardez le PC portable alimenté sur secteur pendant la phase de préparation.

A.3.1.2 Installer les batteries

Le robot est alimenté par 2 batteries « Yuasa NPC17-12 12V 17AH ». Il a été livré avec deux paires de batteries numérotées « set 1 » et « set 2 » (cf. figure A.4).

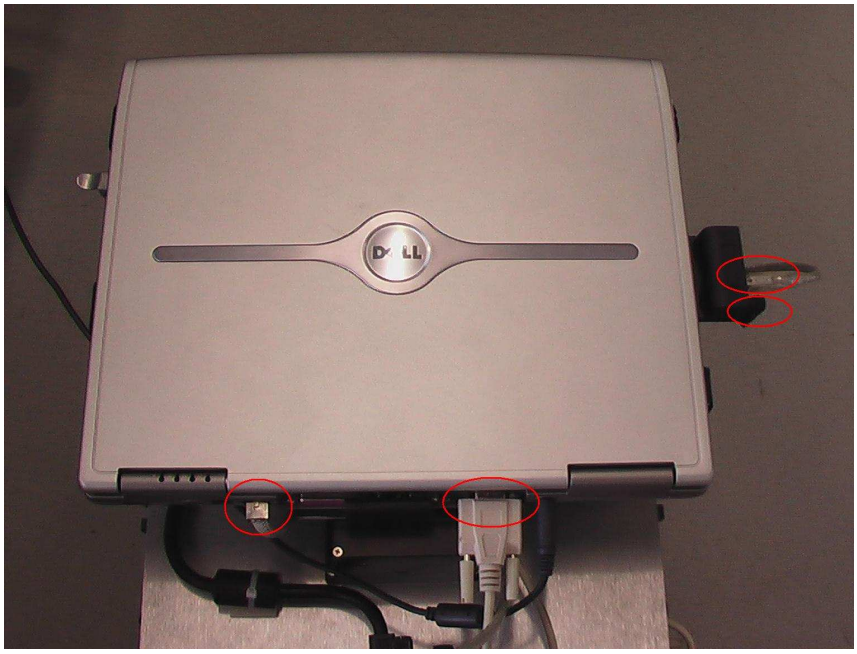


Fig. A.3 : Le PC portable : connexion au robot

Elles doivent impérativement être utilisées et rechargées par paires.
Commencez toujours une démonstration avec des batteries complètement rechargées.

Important : Ne pas mélanger les jeux de batteries, nommés « Set 1 » et « Set 2 ».

A.3.1.3 Préparer le PC embarqué pour le réglage de la fréquence CPU

Le PC embarqué compact PCI est caractérisé par sa fréquence CPU variable – 600 Mhz est sa valeur par défaut. De façon à avoir le maximum de puissance de calcul pour l'inférence bayésienne, *bibabot2* doit fonctionner à sa vitesse maximum : 1.6 Ghz. Ce réglage se fait au démarrage du PC, dans le BIOS. L'initialisation du PC commence à la mise sous tension du robot, il faut donc au préalable brancher un écran et un clavier.

A.3.1.4 Mettre le robot sous tension

A la mise sous tension les différents éléments du robot s'initialisent, entre autres le système pan-tilt, le PowerPC et le PC embarqué.

1. En premier lieu, surveillez que rien ne bloque le mécanisme pan-tilt pour parcourir son espace de mouvement.
2. Ensuite, entrez dans le Bios de *bibabot2* et réglez la fréquence de CPU à 1.6 Ghz.

Fig. A.4 : Ne pas mélanger les sets de batteries

3. Enfin, contrôlez que le PowerPC a bien achevé son initialisation. Celle-ci est terminée lorsque la lampe-flash commence à clignoter puis s'arrête au bout d'une dizaine de secondes. Rappelons que cette phase peut durer quelques minutes, car *bibabot* va télécharger et exécuter son fichier de démarrage.

Le robot est enfin prêt à fonctionner.

Important : Ne pas oublier de régler la fréquence de travail de *bibabot2* à 1.6 Ghz.

A.3.2 Exécution des modules

Une fois la partie matérielle initialisée, la prochaine étape consiste à lancer les différents modules chargés d'exécuter les tâches décrites en section A.2.2 :

1. *La gestion de bas niveau des données capteurs et commandes motrices*,
2. *le traitement vidéo*,
3. *et le contrôle de haut niveau de la sécurité et du comportement du robot.*

Les modules sont recensés sur la figure A.3.2 : en vert, ceux chargés de façon automatique au démarrage du PC et en rouge, ceux qui doivent être lancés manuellement par l'utilisateur.

A.3.2.1 Auto exécution de la gestion de bas niveau

Le module de *gestion de bas niveau des données capteurs et commandes motrices* est programmé pour s'exécuter automatiquement au démarrage du *bibabot*.

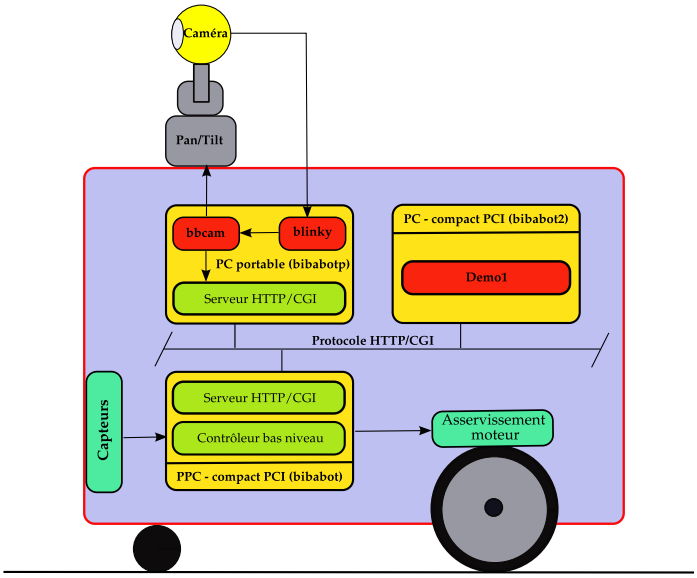


Fig. A.5 : Structure logicielle

Par conséquent, une fois le robot correctement initialisé, ce module est prêt à recevoir des commandes motrices et des demandes de captures sensorielles.

A.3.2.2 Lancer la tâche de traitement vidéo

Le traitement vidéo nécessite le lancement manuel de deux modules :

- **Blinky** : programme d'acquisition vidéo développé au sein de l'équipe Prima de L'Inria Rhônes Alpes.
- **Bibacam** : module de traitement d'images.

Blinky est lui même composé de plusieurs commandes. Nous en retiendrons deux :

- **Blinkyf1394** : également appelé "frontend". Sa principale fonctionnalité est de faire l'acquisition vidéo, via le port firewire.
- **Blinkysdl** : également appelé "backend". Il permet de visualiser la capture. Dans notre application il n'est utilisé qu'à titre de diagnostic.

la commande blinkyf1394 :

Le module **blinkyf1394** est installé sur *bibabotp*, dans le répertoire :

```
$ /usr/local/bin
```

la commande est :

```
$ blinkyf1394 -p<Port no> -c<caméra no> -d<device no> &
```

où :

- *<Port no>* est le numéro du contrôleur firewire sur lequel est connecté la caméra.
- *<Caméra no> est le numéro de la caméra détectée sur le port.*
- *<device no>* est choisi par l'utilisateur.

blinf1394 peut détecter les ports et les caméras disponibles sur le PC. Taper :

```
$ blinkyf1394
```

Le résultat de la commande révèle la présence d'une caméra *IBOT* sur le port 2 (-p2) et a le n°1 (-c1) :

```
$ blinkyf1394
Firewire controllers : 2
Port 1 : 0 cameras
Port 2 : 1 cameras
-----------
Port 2 :
Camera 1 : ORANGE MICRO - IBOT
$
```

La commande à exécuter serait donc :

```
$ blinkyf1394 -p2 -c1 -d1 &
```

Vérifier le bon fonctionnement de la caméra

Premièrement, démarrer le "frontend" de **blinky** avec l'option "'**-s**"' pour commencer la capture :

```
$ blinkyf1394 -p<Port no> -c<caméra no> -d<device no> -s&
```

Deuxièmement, démarrer le visionneur en le connectant au device ouvert par le frontend :

```
$ blinkysdl -d<device no>
```

Une fenêtre devrait afficher la vidéo. Dans le cas contraire vérifier que :

- les paramètres de *blinf1394* sont correctes.
- la caméra est bien connectée,
- qu'elle n'est pas utilisée par un autre programme (ie *coriander*).

Si après toutes ces vérifications le problème persiste, redémarrer la machine.

Une fois la vérification faite, arrêter les deux processus *blinky*.

Important : avant de passer a l'étape suivante, vérifier que les deux processus blinky ne tournent plus.

Lancer la fonction de capture vidéo

Nous allons maintenant démarrer le "frontend" de **blinky**, mais cette fois-ci sans l'option **-s**. Le module de traitement d'image, **bibacam**, demandera à blinky de commencer la capture. Taper :

```
$ blinkyf1394 -p<Port no> -c<caméra no> -d<device no>&
```

Lancer le traitement d'image

Ce module récupère des images de la caméra via **blinky**. Il a pour but de retourner la position et une estimation de la taille d'un objet d'une couleur définie à l'avance. Il se trouve sur le PC portable, dans le répertoire :

```
$ /usr/local/bin
```

Son exécution est simple. Taper :

```
$ bibacam <device no>
```

Une fenêtre devrait afficher la vidéo.

A.3.2.3 Lancer le contrôleur de haut niveau

Cette tâche est à la charge de *bibabot2*, le PC Compact PCI embarqué. Le contrôleur est composé de :

- un script de démarrage : **run**.
- un binaire : **Demo1**.
- un fichier de configuration : *config.xml*.
- plusieurs fichiers d'initialisation du contrôleur de comportements : ***.dat**.
- et de trois librairies dynamiques :
  - **libBBot.so** : interface de bas niveau du robot. Elle exporte l'accès aux données sensorielles et aux commandes motrices.
  - **libBBotOBA.so** : module d'évitement d'obstacles.
  - **libBBotBehavior.so** : contrôleur du comportement du robot.

Le tout se trouve dans le répertoire :

```
$ /usr/local/Demo1
```

Se connecter à distance sur *bibabot2*.
Pour exécuter le contrôleur de haut niveau, il faut tout d'abord se connecter à distance, avec le login **demo** et le mot de passe **demo05** :

```
bibabot2$ ssh demo@bibabot2
```

Une fois connecté, vérifiez dans **/proc/cpuinfo** que la fréquence CPU est de 1.6 Ghz. Si ce n'est pas le cas, branchez un écran et un clavier, redémarrez *bibabot2 uniquement* et réglez la fréquence CPU dans le BIOS. Redémarrez le PC, et reconnectez vous.

Lancer le contrôleur

Pour lancer le contrôleur de haut niveau exécutez la commande :

```
$ /usr/local/Demo1/run
```

Important : avant de lancer le contrôleur, débranchez tout ....

A.4 Arrêter la démonstration

1. Arrêter le contrôleur de haut niveau
Le contrôleur implémente un gestionnaire d'évènements. À la réception du signal SIGKILL (CTRL+C), le contrôleur de comportement termine le cycle en cours, puis s'arrête. Le robot s'immobilise et le pilote de bas niveau est mis en veille.

2. Arrêter le processus de traitement d'image.
Sélectionner la fenêtre du processus de traitement vidéo, *bbcam*, et appuyer sur la touche 'q', la fenêtre se ferme et le programme est stoppé.
Ensuite « tuer » la tâche blinky.

3. Arrêter bibabot2
À partir de *bibabotp* se connecter à *bibabot2* via SSH

4. Mettre hors tension le robot
Basculer simplement le bouton ON/OFF.

5. Mettre les batteries en charge

Bibliographie

[Blu04] BlueBotics. *The BIBA robot : First Steps User Guide.*, 2004. Revision 0047.

[Bon06] Eric Boniface. Gestion d'une plate-forme de vidéo surveillance à usage robotique. modélisation d'un environnement dynamique. Mémoire CNAM, Conservatoire National des Arts et Métiers, Grenoble, France, March 2006.

[CDT05] CDT Community. *CDT User Guide*, March 2005. http ://www.eclipse.org/cdt/.

[Cou03] C. Coué. *Modèle bayésien pour l'analyse multimodale d'environnements dynamiques et encombrés : Application à l'assistance à la conduite en milieu urbain.* PhD thesis, institut National Polytechnique de Grenoble, Grenoble, France, 2003.

[Ecl] Eclipse Foundation. *Extensible development platform and application frameworks.* http ://www.eclipse.org.

[GDB06] GDB committee. *The GNU Project Debugger, GDB Documentation*, January 2006. http ://www.gnu.org/software/gdb/documentation/.

[Gno] Gnome. *Reference Manual for libxml2.* http ://xmlsoft.org/html/index.html.

[Gnu04] Gnuplot. *Gnuplot, An Interactive Plotting Program*, April 2004. http ://www.gnuplot.info/docs/gnuplot.pdf.

[Int03] Intel. *OpenCV Library Reference manual.*, 2003. http ://sourceforge.net/project/showfiles.php ?*group_id* = 22870&*package_id* = 16948.

[Koi05] C. Koike. *Bayesian Approach to Action Selection and Attention Focalisation : An Application in Autonomous Robots programming.* PhD thesis, institut National Polytechnique de Grenoble, Grenoble, France, 2005.

[KPBM03] C. Koike, C. Pradalie, P. Bessiere, and E. Mazer. Proscriptive Bayesian Programming Application for Collision Avoidance. 2003.

[Leb99] O. Lebeltel. *Programmation bayésienne des robots.* PhD thesis, institut National Polytechnique de Grenoble, Grenoble, France, 1999.

[MAK05] N. Mansard, O. Aycard, and C. Koike. Hierarchy of behaviours application to the homing problem in indoor environment. In *IEEE ROBIO 2005, Hong Kong and Macau, China*, July 2005.

[Man03] N. Mansard. Hiérarchie de comportements : application au retour à la base. Mémoire de DEA, institut National Polytechnique de Grenoble, Grenoble, France, June 2003.

NOM : TEIXEIRA PEREIRA
Prénom : Albino
Sujet : Programmation bayésienne de comportements animaux sur un robot mobile

Mémoire d'ingénieur C.N.A.M., Grenoble
soutenu le 28 Juin 2006

Résumé : les systèmes sensori-moteurs autonomes, placés dans un environnement dynamique, doivent en permanence choisir les commandes motrices adéquates à partir des observations obtenues des capteurs. Ce problème est d'autant plus complexe qu'ils doivent faire face aux notions d'incertitude et d'imprécision liés inévitablement aux environnements dynamiques. De plus, l'information obtenue des capteurs peut être bruitée ou manquante, et les commandes motrices imprécises. Les calculs peuvent être approximatifs. D'où l'interrogation : comment percevoir, décider et agir efficacement avec une connaissance incomplète et incertaine ?

L'équipe e-Motion du laboratoire GRAVIR (GRAphics, VIsion and Robotics) a fait de cette problématique un de ses principaux axes de recherche. Elle utilise la méthode et le formalisme de la « Programmation Bayésienne » pour développer des artefacts qui soient capables de traiter des informations incomplètes et incertaines. Les travaux de Carla Koike, membre de l'équipe, ont abouti à un canevas d'implémentation pour contrôler un système sensori-moteur en tenant compte de ces contraintes.

Ce mémoire a pour objectif de présenter l'implémentation d'un comportement animal simple sur un robot en utilisant le canevas proposé, afin de valider qu'il n'est pas seulement théorique, mais qu'il peut être utilisé dans la pratique. Aprés une introduction sur la méthode de la programmation bayésienne et ses principales techniques probabilistes, entre autres le filtre bayésien et la fusion de données, nous présentons la partie théorique du dit canevas. Nous introduisons le robot BIBA, en décrivant ses ressources matérielles et logicielles.

Une fois cette introduction faite, nous détaillons l'implémentation du contrôleur de comportements basé sur le canevas cité ci-dessus et comment il s'interface avec les ressources du robot. Pour garantir la sécurité du robot et de son environnement pendant ses déplacements, nous présentons également l'implémentation un module d'évitement d'obstacles.

Enfin, nous comparons le comportement exécuté et le comportement spécifié pour valider l'implémentation.

Mots clés : Robotique Autonome, Modèle Probabiliste, Fusion Bayésienne, Filtre Bayésien, Sélection de Comportement, Librairie $ProBT^{©}$.

Keywords : Autonomous Robotics, Probabilitic Model, Bayesian Fusion, Bayesian Filter, Behavior Selection, $ProBT^{©}$ Library.