INSTITUT POLYTECHNIQUE DE GRENOBLE

Hiérarchie de Comportements
Application au Retour à la Base

Nicolas Mansard

Encadré par Olivier Aycard

DEA IMAGERIE, VISION ROBOTIQUE

RAPPORT DE STAGE

24 juin 2003

Remerciements

Je tiens tout d'abord à remercier mon maître de stage, Olivier Aycard. En me proposant ce sujet passionnant, il m'a permis à la fois de réaliser des recherches théoriques et dans le même temps de travailler avec des robots, un rêve d'enfance.

Je souhaite également exprimer ma plus vive gratitude aux membres de mon jury pour le temps qu'ils auront passé à travailler, eux aussi, sur ce document. Je voudrais plus spécialement remercier M. Roland Siegwart qui a accepté de venir de Lausanne pour assister à ma soutenance.

Je remercie enfin sincèrement les membres de l'équipe Sharp. En particulier, je remercie les thésards de l'équipe, Carla Koike, Cédric Pradalier, Francis Collas et Christophe Coué, pour leurs multiples conseils et leur aide souvent décisive lors de la programmation du robot.

Table des matières

Remerciements **3**

Table des matières **5**

Chapitre 1. Introduction. **9**

1.1. Situation. *9*

1.2. Problème et problématique. *9*

1.3. Objectifs. *10*

1.4. Contributions et plan de lecture. *10*

Chapitre 2. Environnement pratique et théorique **13**

2.1. Plateforme et lieu d'expérimentation. *13*
2.1.1. Présentation de la plateforme d'expérimentation. 13
2.1.2. L'environnement de travail du robot 14

2.2. Programmation bayésienne. *14*
2.2.1. Quelques rappels. 14
2.2.2. Un exemple simple : évitement d'obstacles « Vrot sachant dir prox ». 15
2.2.3. Pourquoi adopter le formalisme bayésien ? 15

Chapitre 3. Etude bibliographique. **17**

3.1. Méthodes classiques utilisées en robotique. *17*
3.1.1. Représentations métriques. 18
3.1.2. Représentation topologique. 21
3.1.3. Des méthodes mixtes. 22
3.1.4. Conclusion. 24

3.2. L'approche bio-inspirée. *24*
3.2.1. Navigation locale. 25
3.2.2. Recherche d'un chemin. 26
3.2.3. Applications à la robotique : from animals to animats ? 27
3.2.4. Hiérarchiser pour régner. 27

3.3. Synthèse et conclusion. *27*

Chapitre 4. Hiérarchie de comportements. **29**

4.1. Description du comportement. *29*

4.2. Définitions. *30*
4.2.1. Variables sensori-motrices et espace sensori-moteur ({BESSIERE98}). 30
4.2.2. Comportement. 31
4.2.3. Hiérarchie de comportements. 31
4.2.4. Conclusion. 32

4.3. Cohérence d'une hiérarchie de comportements. *32*
4.3.1. Comportement minimal. 32
4.3.2. Cohérence. 33
4.3.3. Conclusion 34

4.4. Formalisation bayésienne. *34*
4.4.1. Séquences temporelles bayésiennes. 35
4.4.2. Relations entre séquence temporelle et hiérarchie de contrôle. 35
4.4.3. Implémentation des hiérarchies de contrôle. 36
4.4.4. Conclusion. 37

4.5. Présentation de notre hiérarchie de contrôle. ... *37*
4.5.1. Une stratégie simple pour rentrer à la base. ... 37
4.5.2. Quatre comportements et deux comportements minimaux. ... 38
4.5.3. Notre hiérarchie de comportement. ... 40

4.6. Spécification pour la suite du travail. ... *40*

Chapitre 5. Briques de base : suivi de couloir. ... **43**

5.1. Présentation. ... *43*
5.1.1. État de l'art. ... 43
5.1.2. Plan du travail effectué. ... 45

5.2. Evitement d'obstacle simple. ... *46*
5.2.1. Présentation. ... 46
5.2.2. Spécification. ... 46
5.2.3. Identification. ... 48
5.2.4. Question. ... 48
5.2.5. Evaluation du comportement. ... 48

5.3. Quatre stratégies pour suivre un couloir. ... *49*
5.3.1. Les quatre stratégies proposées. ... 49
5.3.2. Comportement requis. ... 50
5.3.3. Suite de l'exposé. ... 50

5.4. En construisant un modèle simple de couloir. ... *50*
5.4.1. Présentation. ... 50
5.4.2. Prétraitement des données. ... 51
5.4.3. Programme bayésien. ... 57
5.4.4. Evaluation du comportement. ... 58
5.4.5. Rattachement à la hiérarchie globale ... 60

5.5. En construisant un modèle simple de couloir, seconde version. ... *60*
5.5.1. Présentation. ... 60
5.5.2. Programme bayésien. ... 61
5.5.3. Description et évaluation du comportement. ... 61

5.6. Conclusion et lien avec la suite. ... *62*

Chapitre 6. Briques de base : autres comportements. ... **63**

6.1. Détection d'intersection. ... *63*
6.1.1. Rappel du contexte hiérarchique. ... 63
6.1.2. Stratégie utilisée et variables pertinentes. ... 63

6.2. Tourne droite, tourne gauche … ... *64*

6.3. Suivi d'amer visuel. ... *64*
6.3.1. Rappel du contexte hiérarchique. ... 64
6.3.2. Stratégie utilisée et variables pertinentes. ... 64
6.3.3. Conclusions. ... 65

6.4. « Je suis perdu » : comportement minimal. ... *65*
6.4.1. Rappel du contexte hiérarchique. ... 65
6.4.2. Stratégie utilisée et variables pertinentes. ... 65
6.4.3. Conclusion. ... 66

6.5. Conclusion générale sur les briques de bases. ... *66*

Chapitre 7. Expérimentations et résultats. ... **67**

7.1. Retour à la base. ... *67*

7.2. Partage de cartes et plus court chemin. ... *72*
7.2.1. Partage de carte. ... 72
7.2.2. Plus courts chemins. ... 73

7.3. Conclusion. ... *73*

Chapitre 8. Conclusion. **75**

8.1. Contributions. *75*

8.2. Perspectives. *75*
8.2.1. Développements pratiques. 75
8.2.2. Développements théoriques. 75

Bibliographie. **77**

Annexe A. Exemples de modèles bio-inspirés appliqués à la robotique **83**

Annexe B. Trois comportements minimaux de suivi de couloir. **87**

Annexe C. Guidage dans une intersection **93**

Annexe D. Suivi de direction par amer visuel **101**

Annexe E. « Je suis perdu ! » **105**

Annexe F. Statistiques de réussite des tests. **109**

Chapitre 1.

Introduction.

> … doter le robot mobile des fonctionnalités telles qu'il puisse effectuer de façon efficace, en évoluant d'une manière autonome et sûre dans des environnements inconnus a priori, complexes et dynamiques.
> {NILSSON69}

1.1. Situation.

Le but du projet **BIBA**[1] est de proposer de nouveaux modèles pour la perception, le raisonnement et l'action. Ces modèles s'inspirent des méthodes du vivant pour construire des robots conçus pour le monde réel. Dans le même temps, ils cherchent à comprendre et à expliquer les stratégies des systèmes biologiques, et à tester les hypothèses établies sur des artefacts. Le projet s'inspire des travaux sur l'inférence bayésienne, qui permet une modélisation des dépendances perception – action à l'aide de la logique probabiliste.
Un des travaux effectués dans le cadre de ce projet consiste à construire un robot complètement autonome reproduisant les comportements d'un animal. Entre autres choses, le robot devra être capable de se nourrir seul, de fuir les prédateurs ou encore de recharger ses batteries (l'équivalent de 'boire').
Notre travail porte sur ce dernier comportement. Une des pièces du bâtiment est désignée comme étant la base du robot. A l'intérieur se trouve une borne sur laquelle le robot peut recharger ses batteries. Le but pratique de notre travail est de guider le robot depuis n'importe quel point du bâtiment jusqu'à sa base dans un premier temps, puis jusqu'à la borne de recharge.
Notre problème se ramène donc au *'homing'*, qui fait l'objet de nombreuses études biologiques. Les problèmes de cette classe se résolvent par un guidage utilisant des informations provenant d'un trajet aller. Ces informations sont stockées en mémoire (on parle alors de carte cognitive) ou non (dans ce dernier cas, le mobile doit avoir laissé une piste physique qui lui permette de rentrer: phéromones, mucus, urine ...).
Ce travail se situe donc à la **frontière entre deux disciplines**, la robotique classique d'une part, les études biologiques et biomimétiques de l'autre. La robotique mobile classique a développé de nombreuses techniques permettant à un robot de **construire une carte**, généralement exhaustive, puis de l'utiliser. La **robotique bio-inspirée** expose des méthodes alternatives, en associant à chaque situation rencontrée dans la nature un algorithme, généralement minimal, pour la surmonter.

1.2. Problème et problématique.

Le problème de la *localisation* consiste à déduire des informations des capteurs et d'une carte précédemment mémorisée la position du robot dans son environnement. Ce problème est

[1] BIBA (**B**ayesian **I**nspired **B**rain and **A**rtefacts) est un projet européen dirigé par l'équipe CyberMove du laboratoire GRAVIR, et auquel participent principalement les laboratoires ASL de l'Ecole Polytechnique Fédérale de Lausanne et LPPA du Collège de France.

classiquement étroitement lié à celui de la *navigation*, où le robot utilise les informations obtenues sur la position pour exécuter une trajectoire jusqu'au but.
Nous nous plaçons dans un sous cas de ces deux problèmes : le problème du *homing*. Le robot cherche un chemin pour rejoindre son point de départ en utilisant, pour se localiser, les informations mémorisées pendant son trajet aller. Cette dernière hypothèse permet d'apporter de nombreuses simplifications au problème général de la navigation. En effet, les questions de l'existence d'un chemin ou de la localisation en terrain inconnu ne se posent plus. Dans notre cas, **nous ne nous occuperons que du trajet du retour**. Le trajet aller, en particulier l'acquisition des connaissances nécessaires au retour, est supposé avoir été déjà réalisé.
Plutôt que d'utiliser directement les méthodes classiques de localisation, très lourdes en termes calculatoires, nous utiliserons au maximum les hypothèses qui restreignent notre problème. Nous cherchons des comportements plus légers, et une architecture les liant qui garantisse la résolution du problème. En ce sens, notre problématique peut se formuler ainsi :

« Proposer une architecture de comportements simples permettant de rentrer à la base ».

Nous nous appuierons pour répondre à cette problématique sur les travaux de la robotique bio-inspirée.

1.3. Objectifs.

L'objectif principal de ce travail est de proposer une **architecture bio-inspirée de contrôle** permettant de résoudre le problème du homing en environnement intérieur. Cette architecture doit mettre en œuvre des processus léger nécessitant peu de connaissances. La localisation n'est en effet pas la tâche prioritaire du robot. C'est surtout vrai lors du trajet aller. Le robot effectue une autre tâche qui l'a conduit à sortir de la base. Nous ne voulons pas que la majorité des ressources calculatoires soient utilisées dans la construction d'une carte complexe. Pour cela, nous proposerons une formalisation de la notion de **hiérarchie de comportements**, que nous appliquerons à notre problème en particulier.
Le second objectif de ce travail est de **formaliser cette tâche complexe sous forme bayésienne**. Ce formalisme souple et puissant nous permettra d'exprimer l'ensemble des méthodes et des modèles proposés, tout en y intégrant une gestion des incertitudes.
Le troisième objectif est de **programmer un robot** utilisant les méthodes que nous aurons définies, pour montrer leur validité dans des conditions réelles. Lors du trajet aller, nous supposons que le robot a laissé des marques au sol. Ces marques forment la seule carte nécessaire au robot pour rentrer. La stratégie que nous avons choisie se formule simplement : suivre les couloirs jusqu'aux intersections, et choisir la direction correspondant au nombre maximum de marques détectées.

1.4. Contributions et plan de lecture.

Les contributions de ce travail sont multiples. Tout d'abord, nous avons regroupé et classé lors d'une étude bibliographique de nombreux résultats auxquels la recherche sur les problèmes de localisation, de planification et de cartographie a déjà abouti. Cette étude sera exposée dans le **chapitre 3.** Nous y montrons que ces nombreux travaux offrent déjà des méthodes diverses qui permettraient de résoudre notre problème. Plutôt que prolonger un de ces travaux, nous nous proposons de développer et de formaliser dans cette étude une méthode construite sur une hiérarchie de comportement réactifs, à la suite des études menées dans le cadre bio inspiré.
Dans le **chapitre 4**, nous proposons une formalisation de la notion de **hiérarchie de comportements**. Puis nous montrons comment celle ci peut être mis en oeuvre grâce à la programmation bayésienne, et plus spécialement grâce aux travaux présentés dans

{LEBELTEL99} et {LEHY02}. Nous utilisons alors ce cadre pour construire un ensemble de comportements résolvant notre problème.
Une grande partie de notre travail a été de trouver les comportements répondant en terme d'efficacité aux spécifications du chapitre 3. Nous les présentons dans les **chapitres 5 et 6.** Pour le plus complexe d'entre eux, qui consiste à suivre un couloir en restant au centre, le long de l'axe médian, nous présentons plusieurs versions différentes, toutes conçues dans le même but. Une partie de notre travail a été de faire exécuter cette tâche au robot de manière vraiment satisfaisante, et de trouver des critères permettant de différencier les versions proposées. Nos travaux rejoignent alors ceux effectués dans {YGUEL03}. Les autres comportements seront présentés plus rapidement dans le chapitre 6.
Nous détaillerons alors une série d'expériences validant la méthode hiérarchique proposée. Dans un premier temps, nous présenterons les résultats expérimentaux montrant que le robot effectue bien la tâche prévue. Puis nous montrerons que la programmation choisie aboutit sur des résultats plus forts que prévus. Nous montrerons en particulier que la méthode converge vers un ensemble de plus courts chemins, au fur et à mesure que le robot parcourt son environnement. Cette validation de nos méthodes fera l'objet du **chapitre 7**. Enfin, nous conclurons en nous ramenant aux travaux déjà effectués dans l'équipe sur les cartes bayésiennes.

Chapitre 2.

Environnement pratique et théorique.

Nous présentons ici les bases nécessaires à la compréhension de notre travail. Nous présentons tout d'abord le cadre pratique : le robot et la bâtiment dans lequel il évolue. Puis nous rappelons le cadre théorique des travaux de l'équipe, en reprenant rapidement les principes de la programmation bayésienne des robots.

2.1. Plateforme et lieu d'expérimentation.

2.1.1. Présentation de la plateforme d'expérimentation.

Nous utiliserons pour nos tests expérimentaux le **robot du projet BIBA**, présenté **Figure 2-1**.

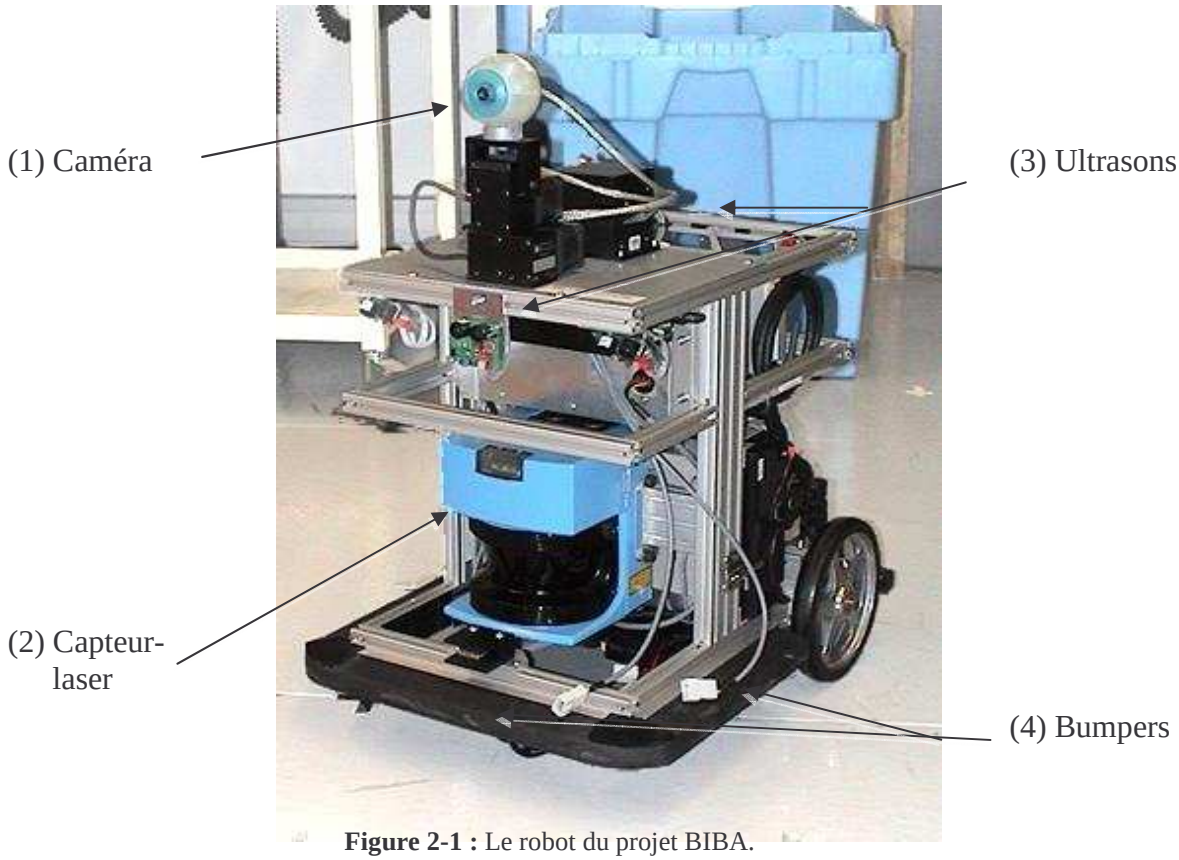


Figure 2-1 : Le robot du projet BIBA.

Il s'agit d'un robot mobile d'environ 50cm de large, de long et de haut. Il est équipé sur l'avant de deux capteurs principaux :

- Un capteur laser (2), qui renvoie un jeu de 361 valeurs couvrant 180° de l'espace. Le laser a une portée maximale de 8m.
- Une caméra pan-tilt (1) renvoyant des images RGB 640×480 à une fréquence maximale de 30 images/seconde.

Le robot est de plus équipé de 'bumpers' (4) qui arrêtent le robot en urgence quand celui-ci touche un obstacle, et de proximètres ultrasons (3) situés sur l'avant et l'arrière du robot. Nous ne les utiliserons pas dans ce travail.
Les ressources systèmes fournissent des ordres permettant de spécifier les valeurs des vitesses de rotation des deux roues arrières séparément (V_g et $V_{d)}$). L'unique autre roue, placée à l'avant, est libre. Nous préfèrerons travailler sur les vitesses de rotation (Vrot) et de translation (Vtrans) du robot. Elles sont liées aux précédentes par les relations :

$$V_g = Vrot + Vtrans$$

$$V_d = - Vrot + Vtrans$$

2.1.2. L'environnement de travail du robot.

Nous avons présenté le robot sur lequel nous testerons les méthodes présentées dans ce travail. Nous allons maintenant présenter le bâtiment dans lequel se déplacera notre robot. Il s'agit des locaux de **l'INRIA** de Montbonnot. Le bâtiment est constitué de couloirs perpendiculaires les uns aux autres, sur lesquels s'ouvrent les portes des différents bureaux. Nous allons recenser ici les particularités qui ont orienté les choix de programmation du robot. Nous reparlerons dans la suite de chacune de ces caractéristiques. Il s'agit ici d'en faire une liste pour que le lecteur ne soit pas surpris lorsque nous y reviendrons.

- Les couloirs se croisent tous à angle droit, en 'X'. Certaines intersections sont plus ouvertes, et du mobilier y est disposé. Le modèle en 'X' ne suffit pas à les décrire correctement. Cela compliquera notre tâche lorsqu'il s'agira de les détecter.
- Les couloirs ne sont pas des murs droits et parallèles. Ils s'élargissent régulièrement devant les portes des bureaux (cf. Figure 5-3). Le robot ne pourra donc pas se contenter de suivre l'un ou l'autre des murs pour avancer dans le couloir.
- Les murs sont peints avec du crépi blanc. Les rayons du capteur laser sont parfaitement renvoyés par cette surface. Les mesures seront donc sûres et précises. Par contre, dans certains couloirs, un des murs est vitré (cf. Figure 5-13). Le robot est incapable de les détecter avec le laser.

Un plan complet du bâtiment est donné sur la Figure 3-1.

2.2. Programmation bayésienne.

2.2.1. Quelques rappels.

Dans cette section, nous rappelons rapidement la méthode de programmation bayésienne introduite dans {LEBELTEL99}. Nous appliquerons cette méthode formelle pour la programmation de notre robot dans toute la suite de ce travail. Cette méthode se décompose en trois phases :

- **Spécification des connaissances préalables**. Le programmeur doit spécifier explicitement les connaissances dont il est à l'origine et celles résultant d'un apprentissage. Ces connaissances sont regroupées en trois parties dans le formalisme utilisé : choix des variables pertinentes, expression des dépendances entre les variables choisies sous la forme d'un produit de distributions élémentaires, et forme paramétrique associé à chaque distributions élémentaires.
- **Identification des paramètres**. Il y a deux manières de spécifier les paramètres des distributions donnés dans la première phase : *a priori*, ou d'après un apprentissage.
- **Question**. Poser une question revient à chercher la distribution de probabilité d'un ensemble de variables *cherchées* (Search) connaissant la valeur des variables *connues*

(Known), en ignorant éventuellement les valeurs des variables du troisième groupe (Unknown). Ces trois ensembles forment une partition de l'ensemble des variables. La question s'écrit :

$$P(Search \mid Known) = \frac{\sum_{Unknown} P(Search, Known, Unknown)}{\sum_{Unknown} \sum_{Search} P(Search, Known, Unknown)}$$

Un vecteur des variables *cherchées* est alors choisi dans l'espace des valeurs possibles. Il peut être par exemple obtenu par tirage aléatoire d'après la distribution question. Nous préférerons choisir la valeur qui maximise la distribution.

2.2.2. Un exemple simple : évitement d'obstacles « Vrot sachant dir prox ».

Nous utiliserons par la suite cet évitement d'obstacle pour nos propres programmes. Nous redonnons ici une version minimale, en guise d'exemple de programmation bayésienne des robot.

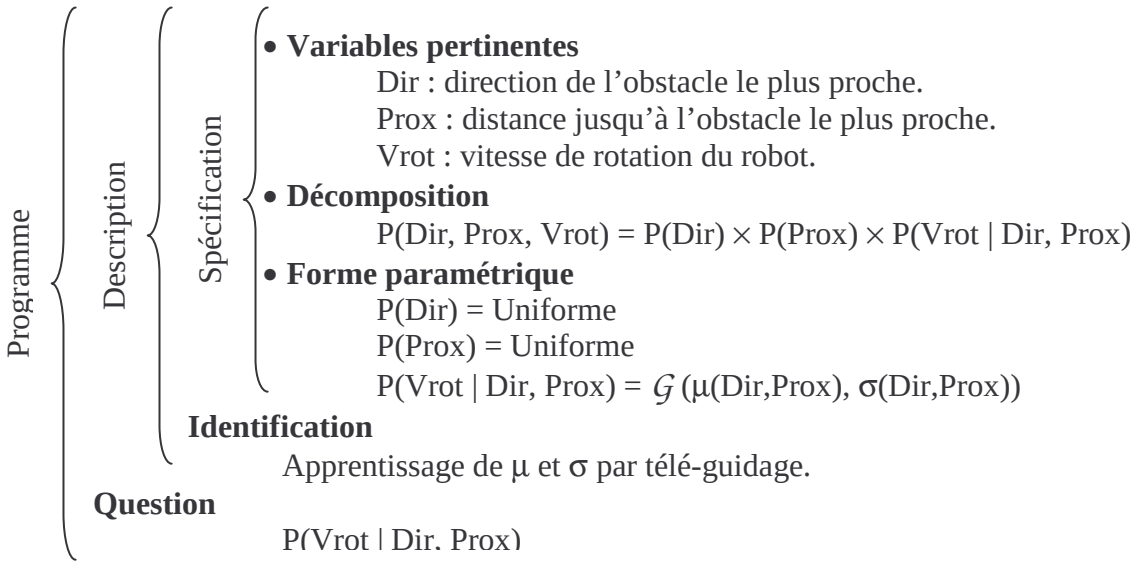


Figure 2-2 : Structure d'un programme bayésien : évitement d'obstacle simple.

Dans cette version, les paramètres μ et σ sont obtenus grâce à un histogramme. Dans l'évitement d'obstacle que nous utiliserons, ils seront fixés *a priori*.

2.2.3. Pourquoi adopter le formalisme bayésien ?

L'approche proposée s'appuie sur des fondements théoriques simples et mathématiquement solides. Ces bases théoriques formalisent d'une part la notion de question, et permettent d'autre part d'y répondre par un raisonnement mathématique. On peut donc l'envisager comme une alternative plausible à la logique en tant que modèle de raisonnement.
La programmation bayésienne permet de plus de raisonner en présence d'incertitude. L'approche développée dans l'équipe CyberMove offre finalement une méthode de programmation et de formalisation simple, systématique et générique.
Nous formaliserons donc tous les algorithmes utilisés dans ce travail en utilisant cette méthode.

Chapitre 3.

Etude bibliographique.

Pour agir de manière autonome, un robot mobile doit être capable de se positionner par rapport à son environnement, de l'appréhender de manière à pouvoir y agir. L'objectif des études sur la localisation est d'apporter des solutions à ces problèmes complexes. C'est pour cette raison que la localisation est souvent présenté comme le premier problème de robotique autonome à résoudre.
La robotique utilise classiquement des méthodes visant à appréhender le plus possible les données extérieures sous forme d'un modèle, puis à se servir de ce modèle pour planifier et prédire la prochaine tâche à accomplir. Mais d'autres réflexions, inspirées des comportements animaliers, proposent des solutions alternatives, qui proposent d'extraire de l'environnement uniquement les données nécessaires à la tâche en cours.
Cette bibliographie se limite à la robotique en environnement structuré. Elle n'a pas la prétention de couvrir les différentes approches pour résoudre le problème de la navigation, au sens large ou même limité au problème du retour à la base qui nous concerne plus particulièrement. En particulier, elle s'attache plus à mettre en valeur les idées fondatrices des différentes méthodes qu'à présenter leur formalisme mathématique ou leurs plus récentes améliorations. Elle compare plutôt ce qui caractérise les différents types de programmations pour permettre au lecteur de comprendre le choix et l'intérêt des solutions que nous proposerons plus tard.
Dans un premier temps, nous étudierons **les méthodes de localisation en robotique classiques**. Deux méthodes sont principalement utilisées : les cartes métriques et les graphes topologiques. Les méthodes les plus récentes proposent de les associer pour obtenir des résultats optimaux. Plusieurs approches différentes dans leur manière de les hiérarchiser seront présentées. Nous exposerons alors les travaux de la robotique bio-inspirée, qui propose une définition de la notion de hiérarchie de plusieurs comportements.
Cette étude bibliographique est construite comme une justification de notre problématique qui, d'une part, propose de chercher une alternative aux méthodes de robotique classique, d'autre par cherche dans la notion de hiérarchie une solution générale permettant de structurer des comportements simples en une stratégie globale susceptible de réaliser une tâche complexe.

3.1. Méthodes classiques utilisées en robotique.

Les méthodes classiques pour résoudre notre problème cherchent généralement à positionner le robot de manière exhaustive par rapport à son environnement. Elles cherchent en fait à répondre aux trois questions introduites en 1990 dans {LEVITT90} :

- où suis-je ?
- où sont les autres lieux par rapport à moi ?
- comment puis-je les rejoindre depuis l'endroit où je suis ?[2]

Même si elles se réfèrent dans cet article au problème de navigation, ces trois questions pourraient s'appliquer presque littéralement à de nombreux problèmes de robotique autonome. Elles sont formulées dans {LEONARD91} d'une manière qui correspond plus spécifiquement au problème de la navigation :

- Où suis-je ?

[2] Notre traduction de: "Where am I? Where are the other places relative to me? How do I get to other place from here?".

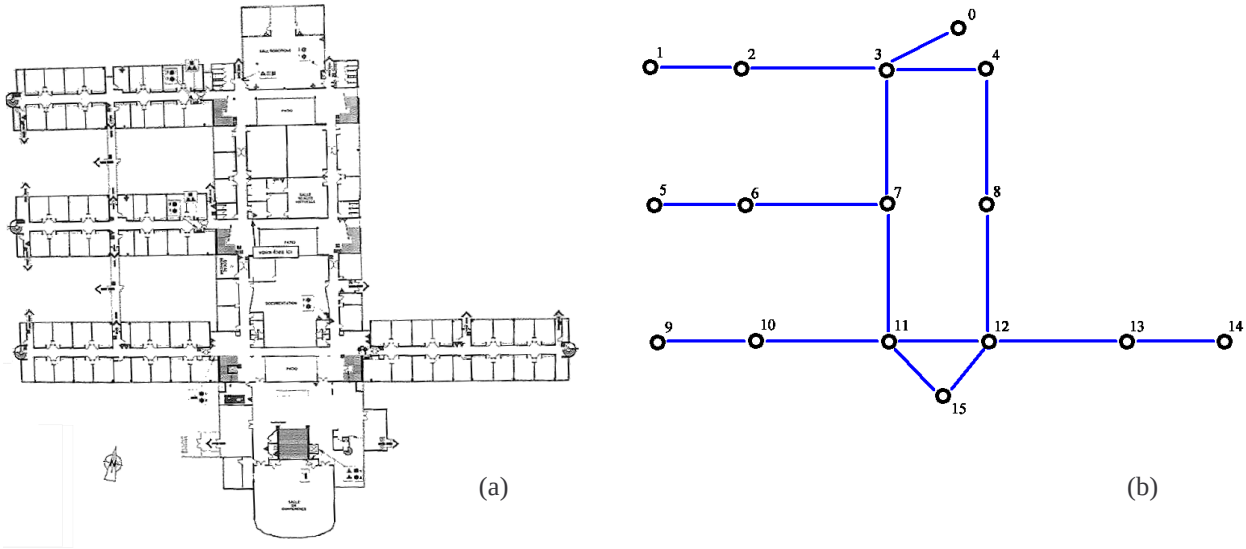


Figure 3-1 : Cartes géométrique (a) et topologique (b) de l'INRIA Rhône-Alpes, bâtiment dans lequel nous effectuerons nos tests.

- Où vais-je ?
- Comment puis-je m'y rendre ?[3]

Le robot se déplace donc en maintenant une représentation aussi exacte que possible de l'environnement. Il cherche parallèlement à se localiser précisément par rapport à cette carte. Les algorithmes de localisation et de modélisation simultanée se regroupent en deux catégories, selon le type de représentations utilisées :

- les représentations métriques : elles sont basées sur une modélisation géométrique de l'environnement (cf. Figure 3-1-a) ;
- les représentations topologiques : elles sont basées sur une modélisation symbolique des lieux (pièces, couloir, etc.) et des relations d'accessibilité entre eux (cf. Figure 3-1-b).

Il faut ajouter que certaines méthodes exploitent parallèlement ces deux représentations, soit en maintenant deux représentations simultanées comme dans {CHATILA85}, soit en se servant de l'une pour construire l'autre comme dans {THRUN98}.

3.1.1. Représentations métriques.

Les solutions présentées dans la littérature sont généralement associées au type de capteur utilisé par le robot. On distingue deux types de capteurs :

- les capteurs extéroceptifs fournissent une mesure de distance entre les objets de l'environnement et le robot. Ce sont des capteurs de distance (laser, ultrason, infrarouge), des cameras stéréo, des radars ...
- les capteurs proprioceptifs fournissent une évaluation de la position du robot, absolue (GPS, boussole ou relative à sa position précédente (odométrie).

La représentation obtenue ressemble fortement à une carte au sens commun du terme. Deux approches pour la construire se sont développées en parallèle : les grilles d'occupation, méthode proposée dans {ELFES89} et les primitives géométriques présentée dans {CHATILA85} et {CROWLEY89}.

[3] Notre traduction de: "Where am I? Where am I going? How should I get there?".

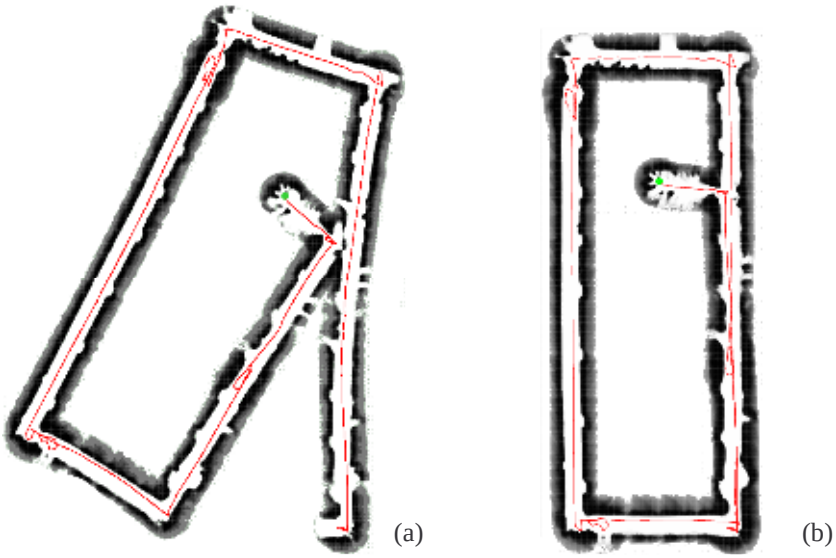


Figure 3-2 : Problèmes du à la localisation par odomètrie lors de la construction d'une grille d'occupation.
(a) La grille d'occupation est tracée en se fiant uniquement aux valeurs de l'odométrie.
(b) Les mêmes données, rectifiées par la méthode de l'E-M
(source: {THRUN97} fig 10a et 10b, page 18).

a) Grille d'occupation.

Ces méthodes discrétisent l'environnement autour du robot en une grille (généralement à deux dimensions, en ne s'intéressant qu'à l'occupation au sol). Les cellules sont disposées régulièrement. A chaque point de la grille est associée la probabilité d'y trouver un objet.
Une formalisation peut être trouvée dans {ELFES89}. La probabilité associée à chaque cellule de la grille y est calculée à l'aide d'un processus bayésien qui intègre les incertitudes liées aux mesures du capteur ultrason. Cette incertitude prise en compte dès le départ dans la formalisation du modèle sous forme probabiliste rend ces méthodes puissantes et robustes. Ainsi, elles sont très bien adaptées à la planification de trajectoire, ou à l'évitement d'obstacle {COUE03}. Elles sont aussi utilisées pour localiser le robot sur une carte connue a priori, en utilisant par exemple un filtrage particulaire (appelé aussi localisation de Monte Carlo) {THRUN00}.
Par contre, elles sont mal adaptées lorsque la compréhension du modèle est nécessaire. Chaque point de la grille est un obstacle, et il n'y a pas de regroupement des différents voisins en un seul objet (c'est par opposition ce qui est fait dans les modèles métriques à base de primitives géométrique). De plus, la définition de la grille est fixée au départ, invariante. On peut augmenter la précision au départ, pour obtenir une localisation plus fine, mais au prix de plus gros calculs. On sera donc toujours limité à une certaine précision décidée au départ. Enfin, un grave problème se pose lorsqu'on cherche à construire une carte de l'environnement inconnu (tâche d'exploration) en réestimant la position du robot grâce à l'odométrie : le cumul des erreurs de positions va déformer la carte, rendant très difficile par exemple le problème de fermer un cycle (cf Figure 3-2).
La localisation utilisant les grilles d'occupation reste donc limitée à l'exécution de tâches robotiques ne nécessitant pas une trop grande précision de localisation, et ni une trop grande compréhension de la sémantique de l'environnement.

b) Primitives géométriques.

L'algorithme se déroule en trois parties. Les primitives géométriques choisies pour la modélisation sont tout d'abord **extraites** des données lues par les capteurs. Puis elles sont **mises en correspondance** avec les primitives déjà extraites et **rassemblées en une carte** de

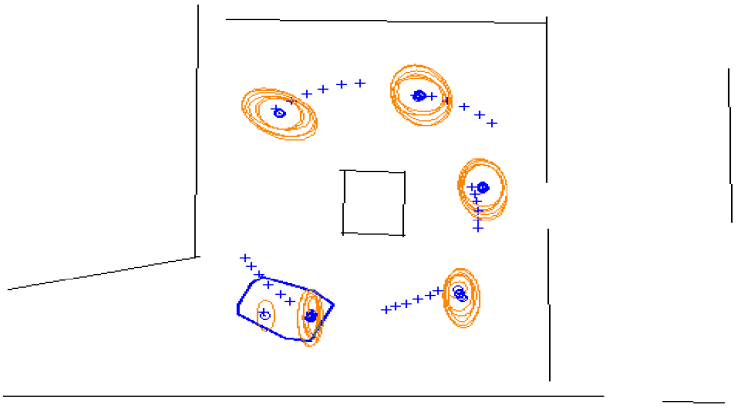


Figure 3-3 : Carte obtenue grâce à une modélisation par primitives géométrique.
Source: {VICTORINO02} fig 6.5 page 144.

l'environnement. Dans {CHATILA85}, les primitives choisies sont des polygones. La carte construite est fusionnée avec les nouvelles parties de polygones découverts par le robot lors de son exploration. Un modèle d'incertitude a été introduit dans {CROWLEY89}, par l'utilisation d'un filtre de Kalman étendu. Les primitives choisies sont les segments de droites, modélisés par leur longueur, la position de leur centre et leur orientation. Une mesure d'incertitude est ajoutée en associant une covariance à chaque segment. La position du robot est elle aussi associée à une covariance. Dans une première phase, le robot se déplace. Une prédiction sur sa nouvelle position est effectuée à l'aide de l'odométrie. Les segments sont alors extraits de la mesure capteur. Ils sont mis en correspondance avec la carte obtenue au temps précédent à l'aide d'une mesure de similarité. Enfin, la position du robot est corrigée à l'aide des informations obtenues lors de la mise en correspondance.
Un exemple de carte obtenue est donné Figure 3-3.

c) Conclusion.

Les différentes méthodes de localisations métriques permettent de créer et de maintenir à jour facilement (en termes de programmation) une représentation très complète de l'environnement. Le robot peut alors interagir avec un environnement qu'il appréhende bien grâce à de nombreuses connaissances.
D'un autre côté, ces représentations sont **très lourdes** car **très complètes**.
Dans les systèmes pionniers, une limitation apparaissaient alors cruellement. Entre deux mises à jour de sa carte, le robot ne pouvait plus interagir avec son environnement. Une nouvelle approche a été proposée dans {BROOKS91} : adopter un comportement réactif. Dans le cadre de cette approche, les actions du robot ne sont plus décidées à l'aide de la carte mais directement à partir des valeurs des capteurs. Un comportement uniquement à base de réflexes déduits des capteurs ne permet évidemment pas de répondre aux mêmes attentes que des décisions construites sur des connaissances aussi complètes que celles apportées par une carte. Un compromis entre un comportement purement réactif et un comportement délibératif comme celui proposé dans {VICTORINO02} est une alternative intéressante : une commande référencée capteur garantit que le robot se situe en permanence sur le graphe de Voronoï de la grille d'occupation (cf. Figure 3-4). Dans un couloir par exemple, cela correspond à se placer au milieu, à même distance des deux murs. Le robot peut ainsi se placer dans son environnement en temps réel, et prendre des décisions plus complexe sur une carte aussi fine que voulue.
Malgré tout, la puissance de calcul nécessaire pour la résolution de certains problèmes, par exemple ceux nécessitant une compréhension élevée de l'environnement à partir des données de la carte, devient trop grande. On a alors besoin d'une autre représentation, plus symbolique, par exemple celle offerte par les graphes topologiques.

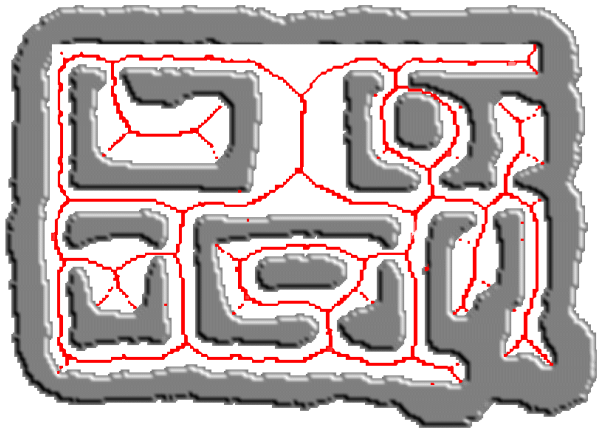


Figure 3-4 : Le graphe de Voronoï est composé de l'ensemble des points à égale distance de deux cellules occupée de la grille.
Source: {THRUN97} fig 14 a, page 27.

3.1.2. Représentation topologique.

Les modèles topologiques sont souvent représentés sous forme d'un graphe :

- les noeuds du graphe sont des lieux, homogènes selon un certain critère ;
- les arêtes représentent les relations d'accessibilité entre ces lieux. On les désigne souvent par chemin.

Contrairement aux cartes métriques, cette modélisation permet une mise en valeur des aspects symboliques de l'environnement ({CHATILA85}). D'une manière générale, les noeuds sont les positions que le robot cherche à atteindre. Des informations sont associées aux noeuds de manière à localiser le robot. Par exemple, dans {JONES97}, le robot peut se situer par rapport aux différents noeuds par une mesure de similarité entre l'image qu'il perçoit à travers la caméra et l'image associée à chaque noeud. De la même manière, les noeuds du graphe présentés dans {FRANZ98} sont indexés par des images panoramiques prises par le robot lors d'un précédent trajet. Le graphe mis en oeuvre n'est plus un simple graphe linéaire comme dans {JONES97}, et permet donc une localisation en tout point de l'univers du robot. De plus, les images panoramiques permettent au robot de se localiser quelque soit le sens dans lequel il arrive sur le noeud. Cet avantage permet en particulier aux auteurs de faire rentrer leur robot à la base !

Les informations contenues dans les noeuds ne permettent pas forcément une localisation unique. C'est par exemple le cas de la carte minimale présentée au chapitre 6 de {DIARD03} : trois types de noeuds sont possibles pour représenter l'arène où évolue le robot : les coins, les bords, et le centre, qui est vide. Le robot détecte très facilement sur quel type de noeud il se trouve (il suffit de compter le nombre de murs proches !), mais cela ne lui permet pas de se localiser de manière absolue dans l'espace topologique. Mais le robot peut se contenter de cette localisation partielle. Elle peut en effet suffire si tout les noeuds sur lesquels le robot estime pouvoir se trouver simultanément conduisent à la même action (c'est le cas dans {DIARD03}). Ou bien il peut chercher à se localiser en intégrant les informations qu'il possède sur la structure du graphe et sur ses précédentes positions, par exemple avec un processus de décision de Markov, comme dans {TAPUS02}.

Les arêtes sont généralement étiquetées par des informations permettant la navigation d'un noeud à l'autre. Dans {KUIPERS89}, les arêtes contiennent la directive à suivre pour rejoindre le noeud suivant (« aller droit », « suivre le mur de droite », « suivre le mur de gauche »). D'autres informations sont référencées : la longueur du chemin à parcourir, une correction des erreurs d'odométrie... Dans {BAILEY01}, les arêtes contiennent la matrice permettant d'effectuer le changement du repère du noeud de départ dans le repère du noeud d'arrivée. Les

coordonnées du robot s'expriment alors facilement par rapport aux amers référencés dans le noeud d'arrivé. Ces points caractéristiques sont utilisés pour corriger les erreurs de position du robot.
Au final, on voit que les connaissances stockées dans une carte topologique sont rangées en une sémantique établie pour faciliter la tâche au robot lors de leurs utilisations. On enregistre par exemple toutes les informations dont le robot aura besoin pour exécuter sa tâche (localisation, ou autre) dans ce noeud. D'autres informations stockées dans les arêtes permettent aux robots de naviguer plus facilement. Une arête peut être un couloir, auquel cas le robot n'a pas vraiment besoin d'informations supplémentaires, mais aussi une porte (ouverte, fermée s'ouvrant dans un certain sens), ou encore un ascenseur ... Le temps de calcul qui était nécessaire au robot pour extraire de la carte les informations pertinentes à chaque position est fortement diminué.
Par contre, la tâche de construction de la carte est rendue d'autant plus difficile. Les informations des capteurs doivent être extraites et stockées. Mais surtout, l'univers du robot doit être découpé en une suite de places importantes, accessible les unes par rapport aux autres.

3.1.3. Des méthodes mixtes.

Les deux méthodes ne sont dénuées ni d'intérêts, ni de désavantages. Un récapitulatif est donné dans {THRUN96}. Nous en donnons ici une version adaptée résumant les deux sous parties précédentes.

Cartes métriques : la résolution ne dépend pas de la complexité de l'environnement.	**Cartes topologiques : la résolution dépend de la complexité de l'environnement.**
(+) facile à construire automatiquement.	(-) difficile à construire automatiquement.
(+) facile à mettre à jour.	(-) difficile à mettre à jour les changements non prévu dans la sémantique.
(-) taille énorme (particulièrement les grilles d'occupation).	(+) taille faible.
(+) permet directement une interaction simple (évitement d'obstacles...).	(-) ne permet pas une interaction directe.
(-) sémantique faible, ne permettant pas directement une compréhension symbolique de l'environnement.	(+) sémantique plus riche.
(+) **permet** une localisation précise du robot sur la carte.	(+) ne nécessite pas une localisation précise du robot.
(-) **nécessite** une localisation précise du robot sur la carte.	

La complémentarité de ces deux méthodes saute aux yeux. Il semble évident qu'en les combinant, on pourra obtenir les avantages de l'un sans les inconvénients de l'autre. Dans {CHATILA85}, des solutions mixtes étaient déjà évoquées. Mais c'est récemment qu'on a pu assister à de vraies mises en application des deux algorithmes couplés.
Dans de nombreux articles (nous en avons présenté certains dans la partie3.1.1.a), Thrun utilisait les grilles d'occupation pour localiser et diriger le robot Rhino dans le musée de Pittsburgh. Dans {THRUN97}, il propose une méthode pour construire un graphe topologique simple (un automate à état fini) à partir de la carte métrique construite dynamiquement. Un exemple des deux cartes obtenues est donné Figure 3-5.
Au contraire, Kuipers et son équipe font partie des pionniers de la localisation utilisant les graphes topologiques. Les limitations présentées par les deux articles {KUIPERS89} et {KUIPERS91} que nous avons vu plus haut trouvent une solution dans {KUIPERS00}.

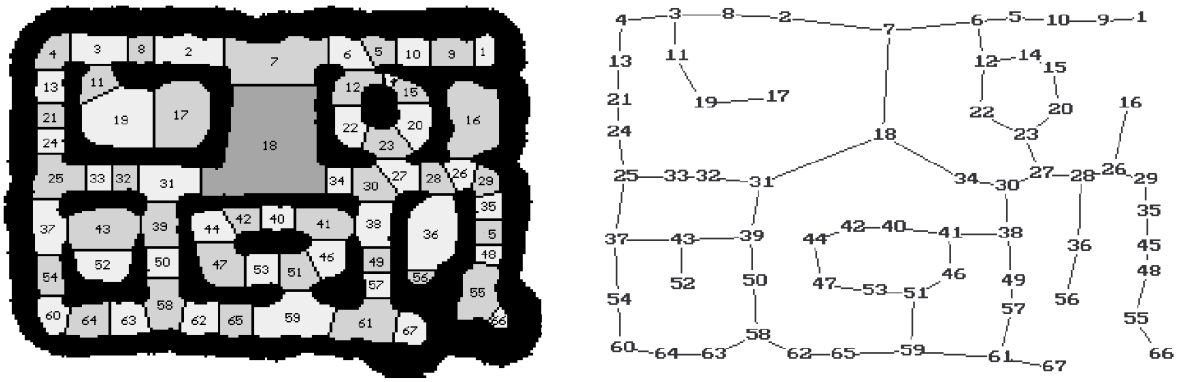


Figure 3-5 : De la carte métrique à la carte topologique.
La carte topologique est obtenue en divisant l'espace libre de la grille d'occupation construite par le robot. Les divisions sont obtenues à partir du graphe de Voronoï repris **Figure 3-4**.
Source: {THRUN97} fig. 14 c et d, page 27.

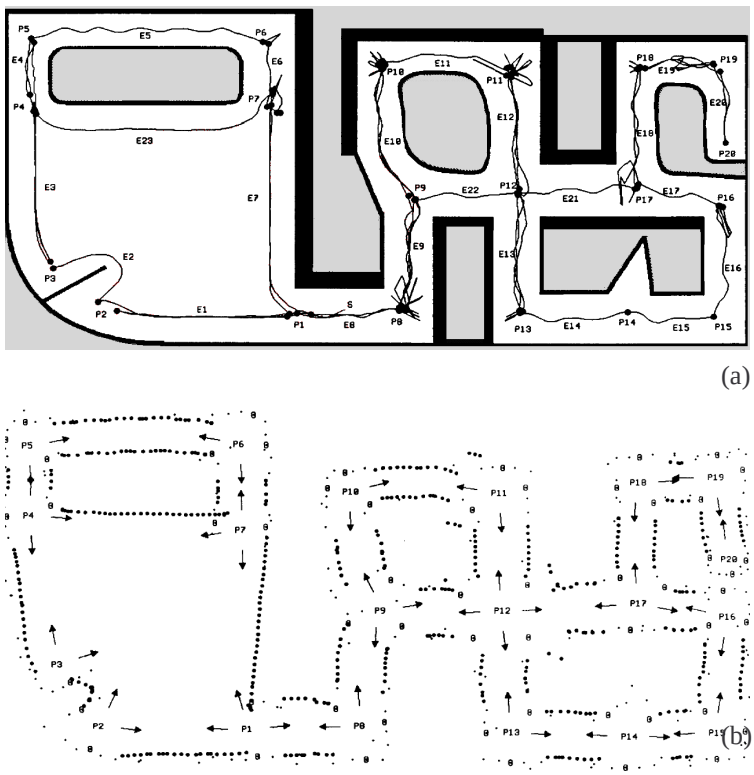


Figure 3-6 : De la carte topologique à la carte métrique.
(a) Un graphe topologique est établi lors du premier trajet du robot.
(b) La carte métrique est obtenue à partir du graphe topologique, et seulement lors d'une seconde exploration.
Source: {KUIPERS97} fig 1a et 1c p. 4.

L'auteur y expose une méthode pour construire la carte métrique de l'environnement après avoir construit un graphe topologique, puis l'avoir renseigné de nombreuses informations métriques. Nous avions vu plus haut que les arêtes de son graphe étaient étiquetées par la distance entre deux noeuds. D'autres informations viennent s'y ajouter lors des passages successifs et permettent au robot d'établir une carte métrique du laboratoire (cf. Figure 3-6).
Une troisième approche est proposée dans {TOMATIS01a} et {TOMATIS01b}. Les auteurs n'utilisent pas l'une des cartes pour construire l'autre mais proposent une méthode hiérarchisant des cartes métriques locales (typiquement des cartes limitées à une pièce) à l'aide d'un graphe topologique. Pour naviguer d'une place à l'autre, leur robot utilise les points caractéristiques répertoriés dans les nœuds du graphe topologique. Un processus de décision markovien est utilisé pour localiser ou relocaliser le robot à l'aide du graphe. Les cartes métriques des lieux du graphe topologique sont utilisés à la fin du trajet pour se guider avec une précision de l'ordre du centimètre jusqu'au but.

3.1.4. Conclusion.

Il y a deux manières de concevoir ces méthodes mixtes : le graphe topologique, symbolique et léger, doit il être construit comme une surcouche de la carte métrique ? Ou bien au contraire faut il utiliser en premier lieu le graphe topologique, généralement mieux adapté à la tâche choisie, et ne se servir de la carte uniquement dans les cas où le graphe n'est plus suffisant, comme par exemple pour le remettre à jour lorsqu'un chemin est coupé, ou pour trouver un raccourci ?
Une fusion des deux méthodes permet d'en améliorer grandement les performances. Mais pour cela, il est nécessaire de **construire une architecture les articulant**. Dans ce cas, on peut se demander pourquoi ne fusionner que deux sources d'informations. Il suffirait de réussir à construire une architecture assez rigoureuse pour pouvoir y intégrer de nombreux comportements différents. C'est sur cette dernière possibilité que nous allons travailler dans la suite. Nous allons donc maintenant nous intéresser aux études robotiques bio-inspirées, qui proposent déjà des critères hiérarchiques entre plusieurs méthodes concurrentes.

3.2. L'approche bio-inspirée.

« Obtenir des robots qu'ils se dirigent de manière autonome s'est avéré être une tâche difficile. Il est donc surprenant que de petits organismes vivants, comme les insectes, aient évolué jusqu'à trouver des solution efficaces à ce problème, malgré un système nerveux simple et des capacités cognitives limitées. »[4]
Cette constatation, qui introduit les méthodes présentées dans {WEBER99}, est l'idée qui sous-tend la démarche bio-inspirée en robotique. Ce type d'approche cherche le plus souvent des stratégies plus simples, moins générales, en reproduisant les algorithmes mis à jour par les biologistes. Le but de cette partie n'est pas de comprendre les mécanismes généraux de localisation du vivant, mais de voir comment les roboticiens ont pu s'en inspirer pour leurs propres travaux.
A notre avis, la plus importante constatation issue des études biologiques est **la hiérarchisation des stratégies utilisées** par les animaux. En effet, s'il est vrai que les insectes, par exemple, utilise des stratégies vraiment simplistes pour se diriger, ils peuvent adopter différents comportements successivement, en fonction de l'estimation de la distance qu'il reste à parcourir jusqu'au but. Et lorsque qu'une méthode complexe ne les amène pas à la solution,

[4] Notre traduction de "Getting robots to navigate autonomously has proven to be a difficult task. It is intriguing, then, that small living organisms such as insects have evolved effective solutions to this problem despite carrying relatively simple nervous systems and restricted processing capacities.", {WEBER99}, premières lignes.

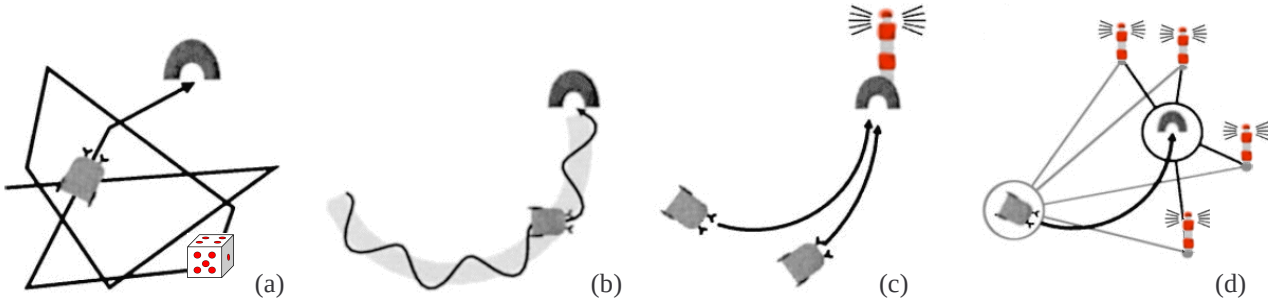


Figure 3-7 : Les quatre types de navigations locales.
(a) Recherche au hasard. (b) Intégration du chemin parcouru.
(c) Suivi d'un amer. (d) Guidage par point de référence.
Source: {FRANZ00}.

ils peuvent passer à une méthode plus simple, qui leur garantit une plus grande chance de réussite (même si c'est souvent au prix d'une perte de temps).
Dans {TRULLIER96}, une distinction est faite entre les différentes méthodes utilisées, selon qu'elles utilisent une représentation 'abstraite' de l'environnement (une carte), aussi simple soit elle, ou qu'elles n'utilisent que les valeurs courantes des différents capteurs pour diriger le mobile[5]. Les auteurs désignent par navigation locale les méthodes de bas niveau, et par comportement de recherche de chemin les méthodes de haut niveau. Ils subdivisent alors les deux groupes en différents sous groupes, et en donnent une hiérarchie complète. Elle est reprise sous une forme légèrement différente dans {FRANZ00}. Nous la redonnons à notre tour ici.

3.2.1. Navigation locale.

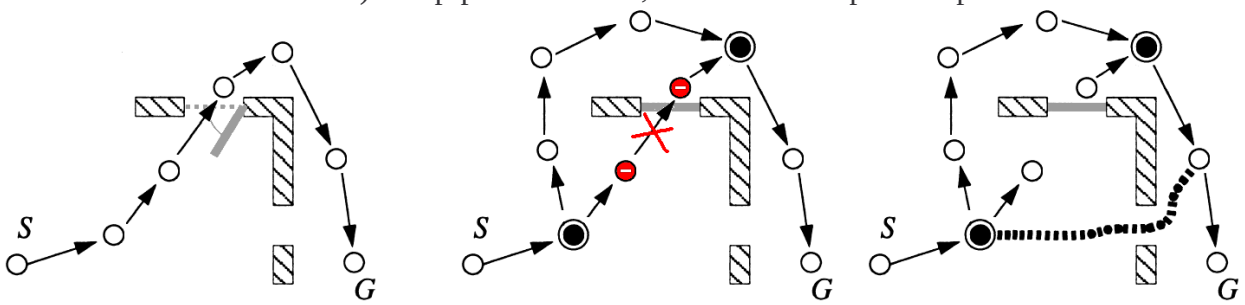
Les méthodes présentées ici sont locales car elles utilisent uniquement les informations fournies par les capteurs pour rejoindre un but. On distingue quatre types de méthodes.

- **recherche au hasard**: les décisions moteur ne sont pas prises dans le but de rejoindre la cible, mais 'au hasard' (Figure 3-7-a). Quand le mobile a atteint la cible, il s'arrête.
- **suivi de trajectoire, et intégration du chemin parcouru**: le mobile connaît la direction à suivre, et mesure la distance qu'il parcourt, à l'aide de capteurs proprioceptifs (odométrie, énergie dépensée) (Figure 3-7-b). Il sait ainsi quand il arrive dans la zone du but, ce qui lui permet de changer de stratégie en cas d'erreur. On est bien dans le cas ou le mobile suit une trajectoire mémorisée (toujours la même).
- **suivi d'un amer**: le mobile se guide en suivant la direction d'un amer qu'il détecte de loin, et corrige ses erreurs de localisation en se recalant en permanence sur l'amer (Figure 3-7-c).
- **guidage par points de référence**: le mobile cherche à retrouver une certaine configuration dans la position de différents points de référence (par exemple se placer « entre les trois cylindres noirs », Figure 3-7-d)

Dans {FRANZ00}, les auteurs précisent qu'un mobile maîtrisant un niveau de stratégie est généralement capable d'exécuter les stratégies des niveaux inférieurs. L'exemple typique est celui d'une fourmi rentrant à la fourmilière ({COLLETT96}). Elle utilise tout d'abord différents repères visibles de loin et entourant l'entrée de la fourmilière. A l'aide d'une image mentale de la manière sont disposés ces repères autour du but, elle s'en rapproche

[5] On parlera dans cette partie de « mobile », pour ne pas faire la distinction entre robot et animal.

({WEBER99}, {MOLLER00}): cela correspond à un guidage par point de référence (l'article est détaillé dans l'annexe A). Trop proche du but, cette méthode perd en précision. La fourmi



(a) (b) (c)

Figure 3-8 : Les trois types de recherches d'un chemin.
(a) Reconnaissance-action : le mobile suit une route sans fourche, de nœud en nœud/
(b) Graphe topologique : si un obstacle bloque une route connu, le mobile peut utiliser un second chemin mémorisé.
(c) Navigation omnisciente : le mobile peut déduire de la carte de nouveaux chemins et des raccourcis en terrain inconnu.
Source: {FRANZ00}.

utilise alors un suivi d'amer, à partir d'un objet plus proche de l'entrée, et plus caractéristique. Finalement, elle peut passer en recherche aléatoire. Les méthodes précédentes l'ayant amené suffisamment proche, cette stratégie pourtant simple garantit un résultat rapide.

3.2.2. Recherche d'un chemin.

Ces méthodes utilisent des connaissances enregistrées lors des trajets précédents pour se guider d'un point connu à un autre.

- **association reconnaissance - action**: le mobile apprend à associer à une image d'un lieu une méthode de navigation locale. On pourrait la décrire mathématiquement par un ensemble de couple, point de départ - méthode de navigation à appliquer. Cette stratégie défini une route, de lieu connu en lieu connu. Si un obstacle empêche le mobile d'appliquer la stratégie courante, ou de rejoindre le lieu connu suivant, il devra changer de méthode (cf. Figure 3-8 a).
- **graphe topologique**: on rejoint à ce niveau les méthodes classiques. La méthode précédente représentait les différentes routes indépendamment les unes des autres. Un but était associé et nécessitait une et une seule carte. Ici, différentes routes sont représentée en même temps. Cela rend la navigation moins dépendante du but choisi. De plus, cela permet de trouver un autre chemin si le premier choisit se trouve bloqué (cf. Figure 3-8-b).
- **navigation omnisciente** : le graphe topologique donne uniquement la position d'une place par rapport à celles qui lui sont accessibles. La navigation omnisciente nécessite au contraire de connaître la localisation de toutes les places les unes par rapport aux autres. C'est typiquement le cas avec une grille d'occupation globale. Cela permet d'inférer un chemin quelconque depuis un point vers un autre. Le mobile peut ainsi découvrir de nouveau chemin, à travers des terrains inconnus (cf. Figure 3-8-c).

On a ici aussi une hiérarchie forte : toutes les informations contenues à un niveau sont incluses au niveau supérieur, et les méthodes d'un niveau correspondent exactement, dans les cas simples, aux méthodes du niveau inférieur.

3.2.3. Applications à la robotique : from animals to animats ?

Les exemples d'applications à la robotique ne manquent pas. De nombreuses équipes ont programmé leur robot de manière à imiter exactement le comportement d'un animal, ou bien en s'en inspirant pour trouver une solution à leurs problèmes. On peut trouver au moins un exemple d'implémentation pour chaque niveau de la hiérarchie. De nombreuses références sont détaillées dans l'annexe A.
Des recherches sont en cours pour essayer de réaliser une architecture de contrôle complexe bio-inspirée. Des robots fonctionnent déjà en utilisant *uniquement* des algorithmes imitant ceux utilisés par les animaux. Sahabot et Sahabot II par exemple, réalisés lors de projets inter-équipe, reproduisent les comportements de la fourmi du Sahara.
Mais aucun robot à notre connaissance n'est suffisamment complet pour réaliser des tâches complexes en autonomie complète, comme le font déjà des robots comme Minerva ou Rhino en utilisant des méthodes de localisation classique (des grilles d'occupation en l'occurrence). C'est l'un des travaux du projet BIBA, dans le cadre duquel nous avons effectué ce travail.

3.2.4. Hiérarchiser pour régner.

Les études bio-inspirées proposent de multiples comportements différents, s'appliquant pour la plupart dans des cas bien particuliers. Mais elles proposent surtout une hiérarchisation de ces différentes méthodes, qui permet d'envisager une généralisation de la fusion entre deux localisations présentée à la fin du paragraphe 3.1.3.

Nous avons présenté les différentes études biomimétiques, en déclinant les différentes classes de comportements, non pas comme exemples de stratégies minimales, mais pour introduire cette notion de hiérarchie. A notre avis, cette conception des comportements débouche directement sur une architecture de contrôle rigoureuse et générale. En la formalisant, il nous a semblé possible d'obtenir des résultats très satisfaisants, en utilisant pourtant des méthodes exploitant peu d'*a priori*, peu de connaissances.

3.3. Synthèse et conclusion.

Rappelons tout d'abord le but de notre travail. Il s'agit de ramener le robot à un lieu qu'il a déjà visité. Le robot s'est rendu lors d'un trajet aller dont nous ne nous occupons pas à sa position actuelle. Lors de l'aller, le robot a pu mémoriser son trajet, ou son environnement. Il doit se servir de ces informations pour rentrer. Il s'agit simplement pour nous de lui faire faire le trajet retour.
Nous pourrions **appliquer les méthodes de robotique classique** que nous avons présentées sans autres problèmes que ceux posés par le développement d'une application aussi complexe que celle requise ici. L'environnement n'est pas extrêmement compliqué. Une méthode utilisant une grille d'occupation s'appliquerait ici à merveille. Il n'y a pas en effet de couloir en boucle qui compliquerait la réalisation de la carte comme sur la Figure 3-2. Le robot pourrait sans problème tracer un graphe topologique dont les noeuds seraient des intersections, et les arêtes des couloirs. Les noeuds seraient indexés par une base de données d'images, ce qui permettrait au robot de se localiser sans erreur.
Mais on peut se demander si **utiliser directement les méthodes de navigation les plus sophistiquées** au sens de la hiérarchie bio-inspirée est utile. On obtiendrait certes sans aucun doute une navigation sûre, mais est-ce la peine d'utiliser autant de ressources de calcul pour une tâche qui pourrait en nécessiter moins ? De plus, cela obligerait le robot à construire une carte complexe lors de son trajet aller, au prix d'une coûteuse consommation de ressource de calcul. Dans notre cas, l'exploration n'est pas la tâche prioritaire. La consommation de

ressource doit être réduite au maximum pour laisser le robot se concentrer sur la tâche qui l'amène à quitter sa base.
Nous allons donc plutôt **nous attacher à construire une architecture de contrôle** utilisant des comportements plus spécifiques, à la manière des comportements bio-inspirés. Aussi souvent que possible, le robot devra utiliser des stratégies simples. Les méthodes de plus hauts niveaux devront être réservée aux cas où le robot ne parvient pas à décider de l'action à entreprendre avec le contrôle de bas niveau. Pour cela, nous allons redéfinir de manière formelle la notion de hiérarchie donnée dans {FRANZ00}, et nous appliquerons cette formalisation à notre problème. Cette structure rigoureuse nous permettra alors d'articuler l'ensemble des comportements nécessaire à l'accomplissement de notre tâche pour obtenir une stratégie globale cohérente.
Cette étude bibliographique nous a permis de détailler nos motivations, et de justifier les choix des méthodes que nous allons maintenant présenter. Dans la suite, nous présenterons tout d'abord les **hiérarchies de comportement**. Cette partie formelle aboutira sur la présentation de l'ensemble des comportements que nous avons programmés pour rentrer à la base, et structure choisie pour les hiérarchiser. Cela nous mènera aux chapitres 5 et 6 où seront **détaillés les comportements un par un**. Le dernier chapitre (chapitre 7) présentera les **résultats expérimentaux** de notre méthode.

Chapitre 4.

Hiérarchie de comportements.

Nous présentons maintenant les solutions que nous proposons pour résoudre le problème du *homing*. Nous avons vu dans la partie précédente que l'état de l'art actuel en robotique mobile offre déjà des solutions suffisamment générales pour s'appliquer à notre cas. Notre approche ne consiste pas à continuer à développer ces techniques, mais à leur proposer une alternative.
Nous désirons programmer notre robot selon une série de processus qui doivent tous être minimaux. Le plus difficile est de **structurer ces « briques de bases »** pour obtenir une méthode globale répondant au problème. Nous allons pour cela proposer une architecture générale, la **hiérarchie de comportements**, et l'appliquer à l'ensemble des stratégies locales que nous aurons établies.
A partir de maintenant, notre travail va se diviser en trois parties. Nous allons tout d'abord **présenter le comportement global** que notre robot va adopter. Celui-ci peut être vu comme un ensemble d'actions articulé sous forme d'automate. Nous nous appuierons plus particulièrement sur les travaux de {LEHY02}, qui présentent une implémentation de ces automates. Notre apport a consisté à **hiérarchiser** ces comportements à la manière des études biomimétiques.
Puis, dans le chapitre suivant, **nous détaillerons chacun de ces états**. Enfin, nous proposerons **une série d'expériences** prouvant que le comportement global est valide, et que la méthode que nous proposons fonctionne.

4.1. Description du comportement.

Programmer un robot à l'aide de comportements simples, structurés à l'aide d'un automate de transition est **une méthode classique**. Le principal problème est alors de déterminer l'état de l'automate à partir des informations en provenance des capteurs. Habituellement, l'automate structure les comportements en offrant une formalisation des transitions entre les états. Mais il ne **les ordonne pas**. D'un autre côté, comme nous l'avons vu dans le chapitre précédent, la robotique bio-inspirée propose, dans le cas du problème de localisation, une **hiérarchie des différentes stratégies** envisageable.
Nous désirons pour notre part un ordre sur les comportements. Le robot appliquant une stratégie à un temps donnée pourra ainsi adapter de manière satisfaisante son comportement quand il détecte que ce dernier ne convient plus à la situation. Si les valeurs capteurs ne correspondent plus au modèle dont il se sert pour déterminer ses ordres moteurs, il passera à une stratégie moins spécifique ('plus basse dans la hiérarchie'). Au contraire, s'il détecte qu'un modèle plus complexe convient mieux, il commencera à utiliser une stratégie plus spécifique ('plus haute dans la hiérarchie'). Nous allons utiliser ces deux notions, d'une part un automate permettant d'implémenter une structure entre différents comportements, d'autre part un ordre sur ces comportements formant une hiérarchie, pour proposer un agencement de stratégies simples s'appliquant à notre problème.
Notre premier objectif a été de donner un cadre formel à la notion de structure de contrôle hiérarchique. Résumons tout d'abord les fonctionnalités que nous désirons obtenir grâce à un tel contrôle.

- **La structure ordonne un ensemble de stratégies** qui permettent au robot de déterminer une valeur pour ses moteurs en fonction des valeurs lues sur ses senseurs.

- A tout instant, le robot doit être capable de déterminer **une et une seule** stratégie pour calculer les valeurs moteur.
- Une valeur est associée à la stratégie courante explicitant **l'intérêt** de ce comportement et la **confiance** à accorder à ce choix. Ces valeurs sont déterminantes dans le choix du comportement à appliquer au temps suivant.
- Si le comportement paraît trop spécifique pour s'appliquer correctement à l'environnement courant, son intérêt baisse. Le robot doit alors passer à un comportement d'un niveau **moins élevé** dans la hiérarchie.
- Au contraire, si un comportement situé **plus haut dans la hiérarchie** paraît s'appliquer à l'environnement détecté, le robot doit adopter cette stratégie.

Dans la partie 3.2.1, nous avions donné un exemple issu de {COLETT96}, présentant les différentes stratégies que la fourmi du désert applique pour rentrer au nid. On peut faire correspondre le guidage qu'elle utilise à un ensemble de comportements respectant les règles que nous venons de donner. L'ordre hiérarchique serait, du plus bas au plus haut: navigation au hasard, suivi d'amer visuel, et guidage par points de référence. Dans un premier temps, le guidage par points de référence est intéressant. Comme il est placé en haut de la hiérarchie, il est choisit comme stratégie courante. Quand le mobile s'approche de son but, la valeur de jugement associée décroît. Il finit par perdre tout intérêt. Le mobile choisit alors le comportement inférieur, suivi d'amer. A son tour, ce comportement perd sa crédibilité. Le mobile utilise alors le comportement le plus simple, recherche au hasard.
Nous allons maintenant nous attacher à **donner des définitions formalisant** les règles que nous venons de donner (§4.2)4.3. Nous montrerons alors que ces définitions correspondent à une structure utile et efficace, que nous désignons par *hiérarchie de comportements* (§4.3).
Nous **ramènerons alors cette structure au cas des automates de comportements classiques**, pour lesquels des implémentations ont déjà été proposées. Ce rattachement à une théorie plus générale déjà bien connue nous permettra de **donner des règles de programmation pour notre structure**. C'est ce que nous ferons dans le paragraphe 4.4.
Dans la dernière partie de ce chapitre (§ 4.5), nous proposerons une utilisation de la hiérarchie que nous aurons définie s'appliquant à notre problème. Il s'agira de diviser la tâche de retour à la base en un ensemble de comportements pertinents, et surtout de définir leur arrangement en hiérarchie.

4.2. Définitions.

4.2.1. Variables sensori-motrices et espace sensori-moteur ({BESSIERE98}).

Définition:

On appelle **variables sensori-motrices** du robot un couple (V_S,V_M) d'ensembles de variables tel que $V_S \cap V_M = \varnothing$.

On notera par la suite N_S = card (V_S) et N_M = card (V_M). On notera de plus N le nombre total de variables: $N = N_S + N_M$.

On appelle **espace senseur** du robot et on note S l'ensemble des valeurs possibles pour les variables de V_S. On a $S \in \Re^{Ns}$.

On appelle **espace moteur** du robot et on note M l'ensemble des valeurs possibles pour les variables de V_M. On a $M \in \Re^{NM}$.

On appelle **espace sensori-moteur** l'espace $E = S \times M \in \Re^N$.

4.2.2. Comportement.

Définition:

On appelle **comportement** sur un espace sensori-moteur S×M un triplet :

$$(\mathbf{f}, \mathbf{C}, \mathbf{I}) \in \mathcal{P}(S\times M) \times (S\rightarrow\{0,1\}) \times (S\rightarrow\{0,1\}\times\Re^{+}).$$

f est un sous ensemble de l'espace sensori-moteur. On peut le voir comme une fonction qui à un vecteur de S associe un vecteur de M. On dit que **f** est la **stratégie** associée au triplet comportement.

C et **I** sont les valeurs qui permettent de juger le comportement. **C** est appelé **crédibilité** du comportement. **I** est appelé **intérêt** du comportement.

Interprétation de la définition.
A chaque instant (c'est à dire à chaque vecteur s ∈ S), **C** associe un booléen. La crédibilité permet de juger si le comportement courant est valide (il l'est si **C** vaut 1 (vrai)).
L'intérêt permet au contraire de juger si les comportements supérieurs sont intéressants (et/ou utiles). **I** associe aux valeurs courantes un couple. La première partie de ce couple vaut soit 0 soit 1 (c'est à dire vrai ou faux). La seconde valeur n'est considérée que si la première vaut 1. Intuitivement, elle permet de comparer deux couples qui sont tous les deux 'vrai'. Si deux comportements supérieurs sont intéressants (**I** vaut vrai pour ces comportements), on décidera d'appliquer celui ayant la seconde valeur la plus forte.
Dans la suite, on dira d'un comportement qu'il est intéressant si **I** vaut vrai, et qu'il est crédible si C vaut vrai.

4.2.3. Hiérarchie de comportements.

Définition:

On appelle **hiérarchie de comportements** un ensemble de comportements **C** muni d'une relation $\mathbf{R} = (\mathbf{R}_{-}, \mathbf{R}_{+})$ telle que :

- $\mathbf{R}_{-}$ et $\mathbf{R}_{+}$ sont deux relations de **C**.
- $\forall (c_1, c_2) \in \mathbf{C}^2$, $(c_1\ \mathbf{R}\ c_2)$ ssi $(c_1\ \mathbf{R}_{-}\ c_2)$ OU $(c_1\ \mathbf{R}_{+}\ c_2)$.
- **R** est **antisymétrique**: $\forall (c_1, c_2) \in \mathbf{C}^2$, $(c_1\ \mathbf{R}\ c_2) \Rightarrow \neg(c_2\ \mathbf{R}\ c_1)$.
- R est acyclique:

$\forall (c, c') \in \mathbf{C}^2$, si $\exists (c_1 \dots c_N) \in \mathbf{C}^N$ tq $c_1\ \mathbf{R}\ c_2, \dots, c_{N-1}\ \mathbf{R}\ c_N$, et $c\ \mathbf{R}\ c1$ et $c_N\ \mathbf{R}\ c'$, alors $\neg(c'\ \mathbf{R}\ c)$.

- R est connexe pour R_{-}:

$\forall (c, c') \in \mathbf{C}^2, \exists (c_1 \dots c_n) \in \mathbf{C}^n$ tq $(c\ \mathbf{R}_{-}\ c_1$ ou $c_1\ \mathbf{R}_{-}\ c)$ et $(c'\ \mathbf{R}_{-}\ c_N$ ou $c_N\ \mathbf{R}_{-}\ c')$ et $\forall i \in \{1..n\}$, $(c_i\ \mathbf{R}_{-}\ c_{i+1}$ ou $c_{i+1}\ \mathbf{R}_{-}\ c_i)$.

- $\forall c \in \mathbf{C}$, si $\exists c' \in \mathbf{C}$ tq $c'\ \mathbf{R}\ c$, alors $\exists!\ c'' \in \mathbf{C}$ tq $(c''\ \mathbf{R}_{-}\ c)$.

Interprétation de cette définition.
Une hiérarchie de comportements peut se représenter par un **graphe orienté** dont les sommets sont les comportements et les arêtes les couples (c_1, c_2) tels que (c_1 **R** c_2). Le graphe obtenu est **acyclique** (d'après la proposition 4). Deux noeuds sont reliés par **au plus une arête** (d'après la proposition 2). Enfin, un noeud à **au plus un seul prédécesseur** au sens de la relation $\mathbf{R}_-$ (d'après la proposition 6).
Une hiérarchie est définie par deux relations :

- On appelle $\mathbf{R}_+$ relation de supériorité. Pour chaque comportement **C**, elle défini les comportements qui peuvent devenir prioritaires par rapport à **C**.
- On appelle $\mathbf{R}_-$ relation d'infériorité. Pour chaque comportement **C**, elle défini le comportement qui doit être appliqué quand **C** devient trop incertain (si la crédibilité devient fausse).

4.2.4. Conclusion.

Ces définitions formalisent les principes que nous avions présentés dans la partie 3.1. Nous allons maintenant chercher à montrer deux propositions. Tout d'abord, nous montrerons que, sous certaines conditions, une hiérarchie de contrôle conduit toujours à appliquer un comportement crédible. Ensuite, nous montrerons qu'une telle structure peut se ramener aux séquences temporelles définies dans {LEBELTEL99}. Cela nous permettra de proposer une méthode d'implémentation, que nous mettrons en œuvre dans la suite.

4.3. Cohérence d'une hiérarchie de comportements.

Notations.
Soient **C** et **C'** deux comportements.
Dans la suite, nous dirons que **C** est inférieur à **C'** ssi **C** $\mathbf{R}_-$ **C'**. De la même manière, nous dirons que **C'** est supérieur à **C** ssi **C** $\mathbf{R}_+$ **C'**. Nous insistons sur la différence de sens entre supérieur et inférieur. 'inférieur' n'est pas le contraire de 'supérieur'. Pour éviter des confusions dans la suite, nous éviterons d'utiliser le terme supérieur, sauf dans un contexte non ambigu.

Nous allons tout d'abord montrer qu'une hiérarchie de contrôle possède un élément 'racine' unique pour la relation $\mathbf{R}_-$. Nous le nommerons comportement minimal. Puis nous montrerons que si le comportement minimal est toujours crédible, alors la hiérarchie ne peut pas conduire le robot à appliquer un comportement incohérent.

4.3.1. Comportement minimal.

Définition :

Soit H une hiérarchie de comportement sur l'ensemble de comportement $\{ C_1 \dots C_N \}$.

On appelle **comportement minimal** de H un comportement n'ayant pas de comportement inférieur, c'est-à-dire tel que :

$$\forall C \in \{ C_1 \dots C_N \}, \neg(C \ \mathbf{R}_- \ C_{min}).$$

Corollaire 1 :

D'après la proposition 3, un comportement minimal n'est supérieur à aucun comportement. On a donc :

$$\forall C \in \{ C_1 \dots C_N \}, \neg(C \ \mathbf{R}_- \ C_{min}).$$

Théorème 1 :

Une hiérarchie de comportement possède un unique comportement minimal. On le note $\mathbf{C_{min}(H)}$**.**

Preuve :
On montre d'abord **l'existence** d'un comportement minimal.
Supposons qu'il n'existe aucun comportement minimal. On va alors construire un cycle pour $\mathbf{R_-}$.
Soit c_1 un comportement de H. On note c_2 l'unique comportement de H qui lui est inférieur. c_2 existe car c_1 n'est pas minimal. On construit ainsi par récurrence sur i une chaîne de N comportements : pour tout i de {2..N}, on note c_i l'unique comportement inférieur de c_{i-1}.
$\forall i \in \{1..N\}$, $c_{i+1} \notin \{ c_k, k=1..i \}$, car sinon, on aurait une chaîne pour la relation $\mathbf{R_-}$. Chacun des c_i est unique. On a donc ordonné l'ensemble C des comportements de H : $C = \{ c_k, k=1..N\}$, car card(C) = N.
On définit maintenant C_{N+1} le comportement inférieur à c_N. $c_{N+1} \in \{ c_k, k=1..N\}$. Soit $i \in \{1..N\}$ tq $c_{N+1}=c_i$. On a construit une chaîne pour $\mathbf{R_-}$ allant de c_i à lui-même, ce qui est en contradiction avec la proposition (6) de la définition d'une hiérarchie. **ABSURDE.**
Un comportement minimal existe dans chaque hiérarchie. Il nous reste à montrer **l'unicité** du comportement minimal.
Supposons qu'il existe deux comportements distincts. On les note c et c'.
Soit $(c_1 \dots c_n) \in \mathbf{C}^n$ tq (**c** $\mathbf{R_-}$ c_1 ou c_1 $\mathbf{R_-}$ **c**) et (**c'** $\mathbf{R_-}$ c_N ou c_N $\mathbf{R_-}$ **c'**)
et $\forall i \in \{1..n\}$, (c_i $\mathbf{R_-}$ c_{i+1} ou c_{i+1} $\mathbf{R_-}$ c_i).
Cette chaîne de comportements existe d'après l'hypothèse de connexité d'une hiérarchie de contrôle.
On a alors **c** **R** c_1, car **c** est minimal (d'après le corollaire 1).
Or, d'après l'hypothèse 6, c_1 ne possède qu'un seul comportement inférieur, c. Donc, c_1 $\mathbf{R_-}$ c_2.
De même, c_2 $\mathbf{R_-}$ c_3 … c_{N-1} $\mathbf{R_-}$ $\mathbf{C_N}$ et c_N $\mathbf{R_-}$ c'. Cette dernière relation est contradictoire avec le fait que C' soit minimal. **ABSURDE.**
Cmin(H) existe et est unique.

4.3.2. Cohérence.

Définition :

Soit H une hiérarchie et c un de ses comportements. On dit que c' est atteignable depuis c pour un état $s \in S$ s'il existe une chaîne de comportements c_1 .. c_N tq :

$c_1 = c$ et $c_N = c'$.

$\forall i \in \{1 ..N-1\}$, c_{i+1} $\mathbf{R_-}$ c_i et $C(c_i)=0$ (c_i n'est pas crédible).

ou (c_i $\mathbf{R_+}$ c_{i+1} et $I(c_{i+1})=(1,m)$ avec $m=\max\{x \text{ tq } ci \ \mathbf{R_-} \ c \text{ et } I(c) = (1,x)\}$).

Définition :

On dit qu'une hiérarchie de contrôle est cohérente si pour tout état de l'espace senseur $s \in S$, et pour tout comportement c, un comportement d'intérêt non nul (c'est-à-dire valant (1,x), $x \in \Re^+$) est atteignable depuis c.

Interprétation de cette définition.
On a défini une structure de contrôle. On veut prouver maintenant qu'elle permet à chaque instant de contrôler le robot de manière raisonnable. Pour cela, il faut que quelque soit l'état des capteurs et l'état précédent, on puisse choisir un comportement crédible au sens donné dans la définition d'un comportement en respectant les deux règles suivantes :

On choisit un comportement supérieur (en passant par une relation $\mathbf{R}_+$) seulement si le nouveau comportement est intéressant (I vaut 1).

Ou bien, on passe au comportement inférieur sans condition.

Ces règles sont spécifiées dans la proposition (2) de la définition d'*atteignable*. Une hiérarchie respectant cette énoncé, formalisé dans la définition précédente, est dite cohérente.

Théorème 2 :

Soit H une hiérarchie de comportements. Soit c_{min} son comportement minimal.

Si est toujours crédible ($\forall s \in S$, $C(c_{min}) = 1$), alors H est cohérente.

Preuve :
Il suffit de montrer que le comportement minimal est toujours atteignable. Soit c_0 un comportement de H et $s \in S$ un état de l'espace senseur.
On construit comme dans la preuve du théorème 1 une chaîne pour la relation $\mathbf{R}_-$ partant de c_0. On la note (c_0 … c_k). On agrandi itérativement cette chaîne en prenant le comportement inférieur à son dernier élément si celui-ci n'est pas crédible. Si on atteint un comportement crédible, on peut s'arrêter : c'est ce qu'on cherchait a atteindre. Sinon, comme la taille de la chaîne est bornée par le nombre de comportements, on finit par atteindre le comportement minimal.
Comme celui-ci est toujours cohérent, on a montré le résultat voulu **:**

H est cohérente.

4.3.3. Conclusion

Nous avons donc défini une structure permettant de choisir une stratégie à adopter à chaque instant parmi un ensemble de comportement. Cette structure est bâtie de manière hiérarchique, ce qui permet de la concevoir comme un automate simple. Nous avons de plus montré que sous une condition simple, la structure garantissait une stratégie cohérente avec les mesures de l'espace senseurs.

4.4. Formalisation bayésienne.

Nous allons maintenant nous attacher à montrer **comment on peut implémenter cette structure** en utilisant les méthodes bayésiennes développées dans {LEBELTEL99} et {LEHY02}. Dans un premier temps, nous rappellerons la définition des **séquences temporelles** introduites dans ces deux travaux (§4.4.1). Puis nous **relierons les hiérarchies de comportements** que nous venons de définir aux séquences temporelles bayésiennes. Il s'agira essentiellement comme nous le verrons de définir les variables capteurs que le robot utilisera pour déterminer le comportement à adopter (§4.4.2). Nous donnerons alors une **méthode pour implémenter notre structure** (§4.4.3).

4.4.1. Séquences temporelles bayésiennes.

Nous redonnons ici rapidement les résultats présentés dans {LEBELTEL99} et {LEHY02}. Le but est de spécifier un programme bayésien général pour exprimer un automate. Nous montrerons ensuite comment utiliser ce formalisme pour notre structure.
Le robot doit utiliser les valeurs de certains de ses capteurs pour déterminer l'état dans lequel il se trouve, avant de prendre une décision pour ces moteurs. On note E_t l'état interne du robot au temps t. On note $\{S_i, i=1 .. n\}$ les variables senseurs pertinentes pour décider de l'état interne du robot. Le but est donc de déterminer l'état du robot en connaissant les variables senseurs et l'état précédent. On obtient le programme bayésien suivant (on ne donne que la partie générale, c'est-à-dire la description et la question).

- **Variables:**

E_t : état courant.
E_{t-1} : état du robot au temps précédent.
S_1 … Sn : variables senseurs pertinentes.

- **Décomposition:**

$$P(E_t, E_{t-1}, S_1, ..., S_n) = P(E_t) \times P(E_{t-1} \mid E_t) \times \prod_{i=1}^{n} P(S_i \mid E_t)$$

- **Forme paramétrique:**

$P(E_t)$: on ne fixe pas d'a priori sur l'état courant. On utilise donc la loi uniforme.
$P(E_{t-1} \mid Et)$, $P(S_i \mid E_t)$: on spécifie ces lois par des tables de probabilité. Ces lois peuvent être données par le programmateur, ou apprises. Nous choisirons la première solution

- **Question.**

$$P(E_t \mid E_{t-1}, S_1, \ldots, S_n).$$

Cette formalisation bayésienne nous permet d'exprimer un graphe de comportements. Il nous reste à relier la définition d'une hiérarchie de contrôle à cette formalisation.

4.4.2. Relations entre séquence temporelle et hiérarchie de contrôle.

Il existe donc une méthode pour passer d'un automate de comportements à un programme bayésien. Nous allons voir ici comme l'appliquer aux hiérarchies de comportements. Il s'agit de **déterminer** d'une part ce qu'est **un état** (noté E_t pour l'automate bayésien) dans une hiérarchie de comportements, d'autre part quelles variables sont pertinentes lors de la décision de l'état courant. Cela revient, en utilisant les notations ci-dessus, à **choisir les S_i**.
Une hiérarchie de comportements est un automate dont les transitions sont de deux types bien distincts. Les états de l'automate sont bien évidement les comportements de la hiérarchie. Les transitions sont choisies en fonction des valeurs de crédibilité et d'intérêt des comportements. L'évolution du comportement courant est décidée d'après trois règles que nous avions évoquées lors de la définition de atteignable (§4.3.2) :

- On choisit un état supérieur à l'état courant si et seulement si sa valeur d'intérêt vaut (1,m), m étant le maximum des indices d'intérêt des états supérieurs.
- Sinon, si l'état courant est crédible (C vaut 1), on y reste.
- Sinon, on passe d'état inférieur à état inférieur jusqu'à trouver un comportement crédible.

Pour résumer, on passe dans l'état supérieur le plus intéressant s'il existe, sinon on reste dans l'état courant ou on passe à l'état inférieur selon la crédibilité courante. Les variables capteurs pertinentes sont donc les variables C et I des comportements de la hiérarchie.

L'utilisation des séquences temporelles bayésiennes va nous permettre de proposer une implémentation pour les hiérarchie de contrôle sans les redéfinir explicitement comme des automates. En effet, comme nous l'avons vu dans le paragraphe suivant, il nous suffit de spécifier les tables de probabilités en respectant les règles que nous venons d'énoncer. C'est ce que nous allons faire dans le prochain paragraphe.

4.4.3. Implémentation des hiérarchies de contrôle.

Nous avons défini l'ensemble des valeurs que peut prendre la variable d'état. Nous avons aussi défini notre ensemble de variables senseurs. Il y en a 2×N, avec N le nombre de comportements (pour chaque comportement, on a I et C). On peut donc utiliser directement le programme bayésien défini dans le paragraphe 4.4.1. Il ne nous reste plus **qu'à spécifier les tables de probabilités** pour $P(E_{t-1} \mid E_t)$et pour les $P(S_i \mid E_t)$.
Dans la suite, nous considérerons une hiérarchie de N comportements. Nous notons <i> le $i^{ème}$ comportement. Nous notons C_i et I_i les valeurs de crédibilité et d'intérêt qui lui sont associés. On donnera les tables de probabilité sous forme d'histogrammes non normalisés, pour une meilleure lisibilité. Les valeurs de probabilités données devront donc être comparées aux autres valeurs du tableau pour avoir un sens.

a) P(Et-1 | Et).

La première table correspond aux probabilités de transitions entre deux états. Elle est construite pour stabiliser les états (équation (1)), et pour respecter la relation **R**. Il s'agit simplement d'une part d'interdire les passages entre deux comportements qui ne sont pas en relation, d'autre part de spécifier quelles importances relatives on donne aux relations $\mathbf{R}_+$ et $\mathbf{R}_-$.

La première équation privilégie la stabilité du système :

$$\forall i \in \{1..N\}, \quad P([E_{t-1} = <i>] \mid [E_t = <i>]) = 5.$$

On interdit les passages entre deux états qui ne sont pas reliés par une flèche, et on autorise les passages entre deux états voisins. On privilégie les passages vers des états supérieurs plutôt qu'inférieurs. On obtient la table suivante :

$\forall (i,j) \in \{1..N\}^2$, tels que $i \neq j$:

- $P([E_{t-1}=<j>] \mid [E_t=<i>]) = 2$, si <i> $\mathbf{R}_+$ <j> **;**
- $P([E_{t-1}=<j>] \mid [E_t=<i>) = 1/l$, si <j> $\mathbf{R}_-^*$ <i> ;
- $P([E_{t-1}=<j>] \mid [E_t=<i>]) = 0$ sinon.

On note <j> $\mathbf{R}_-^*$ <i> s'il existe un chemin de relation $\mathbf{R}_-$ entre les comportements i et j. Dans ce cas, on note l la longueur du plus court chemin de relation $\mathbf{R}_-$ de <j> à <i>**.**

b) P(Ci | Et) : variables de crédibilité.

Cette table montre comment mettre en relation une variable et le comportement choisi. On se contente t'interdire le comportement i quand sa crédibilité C_i vaut 0 (6), et d'en augmenter la probabilité quand elle vaut 1 (7) :

$\forall j \in \{1..N\}, j \neq i$:

- $P([C_j=0] \mid [E_t=<i>]) = 1.$
- $P([C_i=0] \mid [E_t=<i>]) = 0.$

c) P([Ci=1] | [Et=Ci]) = 2.P(Ii | Et) : variable d'intérêt..

Nous avons défini I comme un couple formé d'un booléen et d'un réel positif. Nous avions alors précisé que le réel ne nous intéressait que quand le booléen était vrai, comme quantification de cette valeur vraie. On décide de ne représenter que le booléen par la variable I_i. On note x_i la valeur du réel. Nous allons nous en servir pour spécifier la table de vérité.

$\forall j \in \{1..N\}, j \neq i$:

- $P([I_j=0] \mid [E_t=<i>]) = 1.$
- $P([I_i=0] \mid [E_t=<i>]) = 1.$
- $P([I_i=1] \mid [E_t=<i>]) = x_i.$

On note que si $x_i < 1$, la transition vers C_i est moins probable que si I vaut 0.

4.4.4. Conclusion.

A partir des définitions données en 4.2, nous avons défini un programme bayésien nous permettant de hiérarchiser un ensemble de comportements. Nous n'avons pas montré que ce programme respecte les spécifications données pour une hiérarchie de comportements. Ce n'est d'ailleurs pas exactement le cas. Les transitions définies dans les tables de probabilités pour l'automate bayésien sont moins strictes que celle de notre réseau hiérarchique. En particulier, on garantissait avec la hiérarchie qu'aucun comportement supérieur ne serait pas choisit s'il n'était pas intéressant. Ici, cela ne suffira pas. Il faudra en plus avoir une crédible à 0 pour que la probabilité d'être choisi soit nulle.
On préserve par contre la propriété donnée par le théorème 2 : un état crédible est atteignable à chaque instant, à condition que le comportement minimal soit toujours crédible.

4.5. Présentation de notre hiérarchie de contrôle.

Nous allons maintenant présenter une application de toutes les spécifications que nous venons de donner. Rappelons notre objectif : il s'agit de rejoindre une pièce spécifique du bâtiment en se servant connaissances acquise lors d'un trajet aller.

4.5.1. Une stratégie simple pour rentrer à la base.

Nous avons décidé d'utiliser une stratégie limitant au maximum les connaissances *a priori* nécessaires au robot. Pour cela, nous allons imiter le comportement adopté par les fourmis marquant leur trajectoire à l'aide de phéromones. Nous n'avons pas de détecteur de phéromone. Par contre, le robot est équipe d'un distributeur de boulons[6]. Ceux-ci, peints en rouge, sont utilisés par le robot pour marquer le couloir lors de son arrivée à une intersection, pendant le trajet aller. Un processus très simple permet de détecter lors de l'aller les endroits où poser les marques. Ces marques rouges au sol constitueront les seuls *a priori* que le robot utilisera lors du trajet retour. On considère de plus que le nombre de boulons dans la dernière intersection est supérieur à cinq.

[6] En réalité, le distributeur de boulons n'est pour l'instant qu'un prototype. Nous avons utilisé des carrés de papier de couleur rouge, placés à la main sur le sol avant le passage du robot.

Il ne reste plus au robot qu'à **longer le couloir** jusqu'à arriver à une intersection. Le premier comportement consiste à suivre le couloir. Quand le robot arrive à une intersection, l'**intérêt du second comportement, 'détection d'intersection', augmente** et le robot passe à une stratégie plus élevée dans la hiérarchie.
Il choisit la direction **du couloir ayant le plus de marques rouges au sol**. Il tourne alors dans cette direction Si le nombre de marques est inférieur à cinq, il repasse en comportement 'suivi de couloir'.
Lorsque le nombre de marques détecté est suffisant (supérieur de cinq), on considère qu'il est arrivé à proximité de la base. Il utilise alors le dernier comportement, et rejoint la base en se guidant sur **un amer visuel**. L'ensemble de ces comportements est représenté sur la Figure 4-1.

4.5.2. Quatre comportements et deux comportements minimaux.

Nous avons donc besoin de quatre comportements :

- Suivre le couloir.
- Choisir un des quatre couloirs possibles dans une intersection.
- S'orienter dans la direction donnée par le comportement précédent.
- Rejoindre la base une fois que le robot est arrivé à proximité.

On définit aussi un comportement minimal. Quand le robot l'adopte, il se met à tourner sur lui-même. Nous en montrerons l'intérêt dans le paragraphe 6.4. Nous définissons de plus un dernier comportement. Il s'agit pour le robot de suivre la direction la plus lointaine donnée par le capteur laser. Cela le conduit à suivre le couloir, mais de manière moins précise que le comportement suivi de couloir. Par contre, ce comportement moins spécifique permet au robot de trouver une solution à d'autres problèmes que le seul suivi de couloir. En particulier, le robot arrive grâce à cela à sortir d'une pièce où il est entré par erreur.
Nous détaillerons les stratégies de ces comportements dans le chapitre 4. Nous allons détailler ici les valeurs de crédibilité (C) et d'intérêt (I).

a) Suivre le couloir.

Les décisions motrices sont prises en fonction d'un modèle du couloir que le robot reconstruit à chaque instant. Nous verrons dans le chapitre suivant les différentes caractéristiques du couloir que nous avons utilisé pour construire ce modèle : distance au mur, direction des mesures laser les plus lointaines... A chacun de ces modèles, on peut associer une valeur en jugeant la certitude. On la note p, comprise entre 0 et 1. Cette valeur sera détaillé pour chaque comportement dans le chapitre 5. On définit C et I à l'aide de p :

$$C = (p > 30\%).$$

$$I = ((p > 50\%), 10 \times p).$$

b) Guidage dans une intersection : choisir un des quatre couloirs.

Nous associons à chaque jeu de mesures du laser une valeur de probabilité d'avoir détecté un couloir perpendiculaire à la direction que le robot suit. On la note q, comprise entre 0 et 1. Nous détaillerons le choix de cul dans le chapitre 6. On spécifie I en fonction de q :

$$I = ((p > 30\%), 10 \times p).$$

Par contre, une fois que le robot à compté les marques rouges au sol (ce qui lui prend un temps de calcul), il faut qu'il change de comportements. On fixe sa crédibilité à 0.

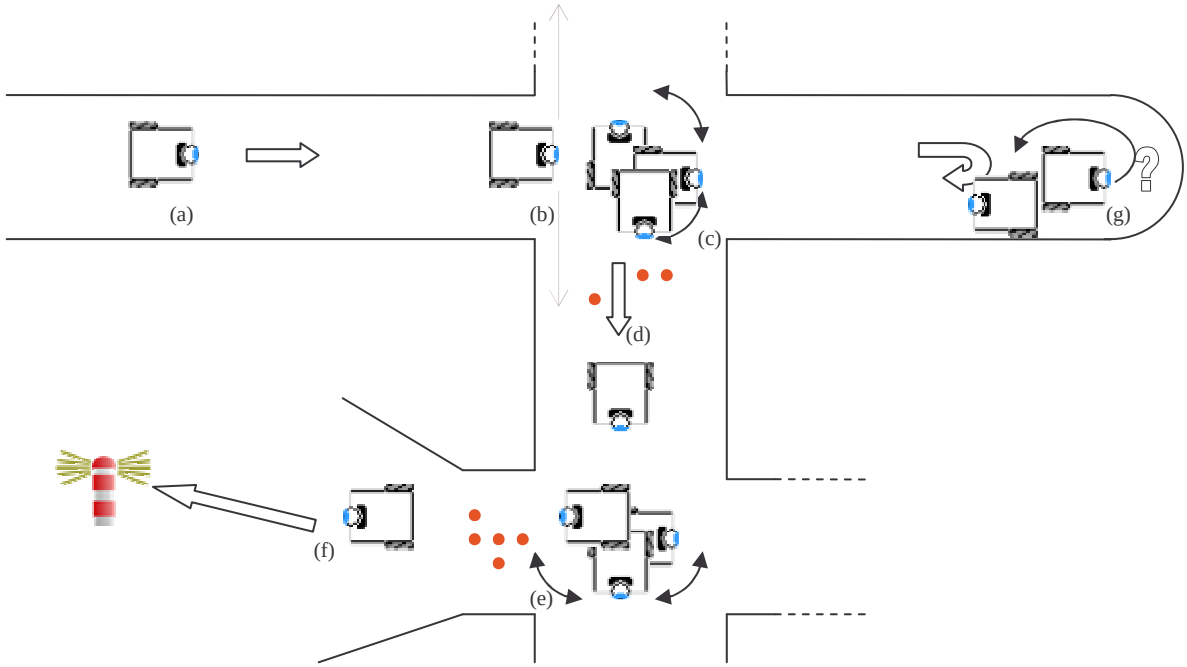


Figure 4-1 : Les différents comportements adoptés pour retourner à la base :
(a) Suivi de couloir.
(b) Le robot détecte une intersection.
(c) Recherche dans les différents couloirs possibles des marques rouges au sol.
(d) Le robot choisi de tourner à gauche.
(e) Cinq boulons rouges au sol : le robot est assez proche de la base pour changer de comportement.
(f) Suivi d'amer jusqu'à la base.
(g) Au bout du couloir, le robot adopte le comportement minimal et fait demi-tour.

c) S'orienter dans la direction des marques.

On définit ici quatre comportements : tourner à droite, à gauche, aller tout droit et faire demi-tour. Leur intérêt est donné en fonction du nombre de marques rouges détectées au temps précédent.

$$\forall i \in \{\text{droite, gauche, tout\_droit, demi\_tour}\},$$

$$I_i = (\,(\text{marque}(i, t\text{-}1) \neq 0), \text{marque}(i, t\text{-}1)).$$

Comme pour le comportement précédent, on veut que ce comportement s'arrête dès qu'il a été exécuté un cycle. On fixe donc C à 0.

d) Rejoindre la base une fois que le robot est arrivé à proximité.

Ici aussi, on peut définir quatre comportements, chacun donnant une direction où chercher la base. On utilise la même fonction pour calculer I que pour le comportement précédent, en imposant que le nombre de marques soit plus grand que cinq :

$$\forall i \in \{\text{droite, gauche, tout\_droit, demi\_tour}\},$$

$$I_i = (\,(\text{marque}(i, t\text{-}1) > 0), \text{marque}(i, t\text{-}1)).$$

Par contre, nous ne voulons pas que ce comportement cesse dès qu'il a commencé. Nous fixons un critère de crédibilité en se servant du temps. La crédibilité décroît donc strictement à chaque pas de temps, jusqu'à ce que le comportement soit jugé non crédible. Cela signifiera que le robot a fait une erreur en cherchant la base dans cette direction. Il change alors de stratégie et

cherche à rejoindre un couloir. On note T_{max} la durée à l'issue de laquelle le robot doit changer de comportement.

e) Direction la plus lointaine.

Ce comportement joue finalement le même rôle que le suivi de couloir dans la hiérarchie avec moins d'*a priori* et de moins bons résultats. On note d la plus grande distance renvoyé par le capteur laser au temps courant, et d_{max} la valeur maximale que peut renvoyer le laser.

$$C = (\frac{d}{d_{max}} > 30\%).$$

$$I = ((\frac{d}{d_{max}} > 50\%), 10 \times \frac{d}{d_{max}}).$$

4.5.3. Notre hiérarchie de comportement.

Le plus simple est de représenter la hiérarchie structurant ces comportements par son graphe. Il est donné Figure 4-2. Les relations $\mathbf{R}_+$ sont données en noir. Les relations $\mathbf{R}_-$ sont en rouge. Les valeurs correspondantes C et I sont données sur la droite du schéma.
On retrouve bien quatre étages dans la hiérarchie. Nous avons divisé dans la pratique en quatre le comportement consistant à se tourner dans la direction correspondant au maximum de marques rouges. Les deux comportements inférieur, suivi de direction lointaine et le comportement minimal sont désignés par le même ensemble : « Je suis perdu ».

4.6. Spécification pour la suite du travail.

La suite de ce travail se déduit directement de ce que nous venons d'expliquer. Il nous reste pour finir de présenter la programmation du robot à détailler chaque comportement de notre hiérarchie. Nous venons de spécifier les valeurs d'intérêt (I) et de crédibilité (C) de ces comportements. Il faut maintenant expliciter les stratégies associées. C'est ce que nous allons faire dans les chapitres 4 et 5.
La plus grosse partie exposera en détail le comportement « suivi de couloir ». Ce sera le sujet du chapitre 4. Nous nous attarderons moins sur les autres, qui seront résumés dans le chapitre 5. Ces comportements, dont l'intérêt innovateur est moindre, sont détaillés en annexe C, D et E. Nous rappelons ici que nous continuerons d'utiliser pour ces spécifications le même formalisme bayésien comme nous l'avons fait jusqu'à présent.

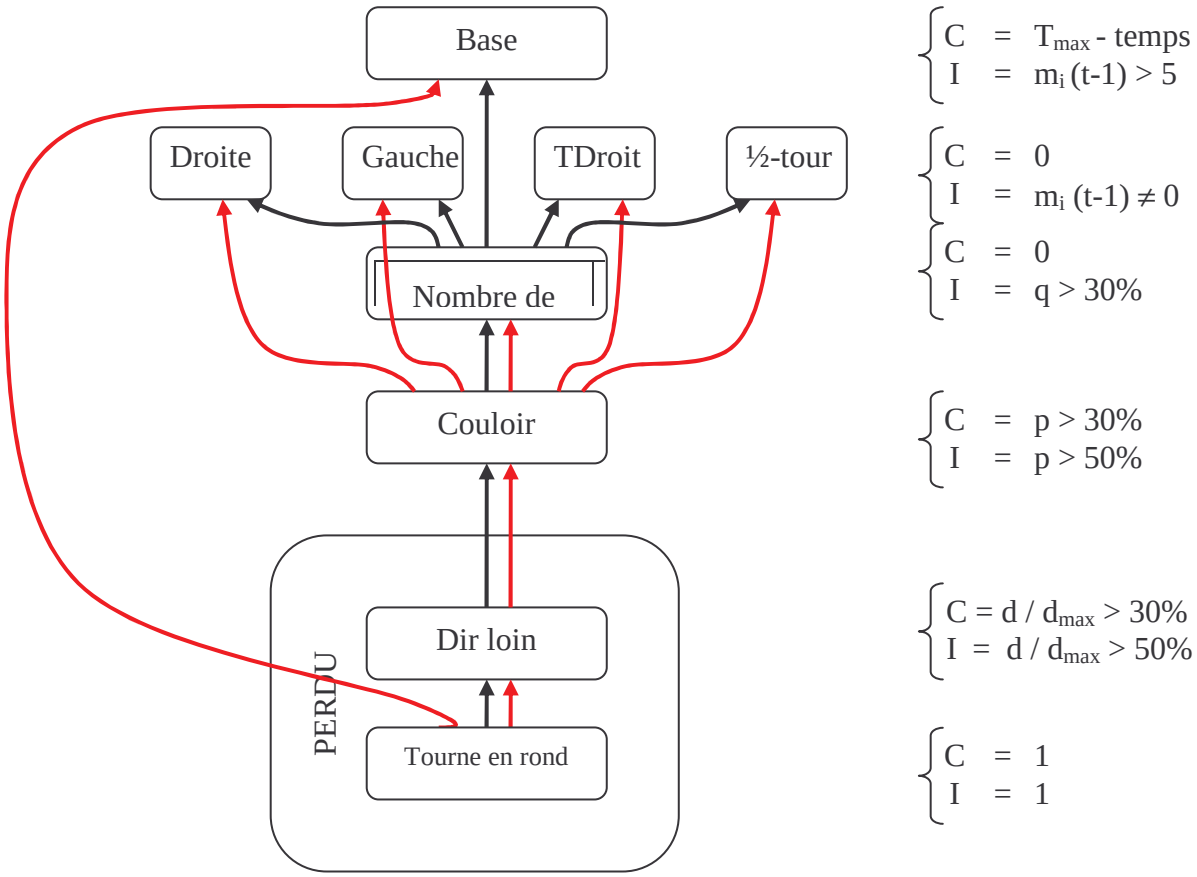


Figure 4-2 : Graphe de la hiérarchie de comportements utilisée pour diriger le robot.
Le comportement minimal est « tourne en rond ».
- « dir loin » permet un suivi de couloir moins fin, mais plus général que le suivi de couloir.
- « couloir » consiste à longer le couloir. C'est le comportement de navigation locale le plus important de la hiérarchie.
- « intersection » est utilisé quand une intersection est détecté.
- Selon la direction où se situe le maximum de marques rouges dans l'intersection, un des quatre comportements « gacuhe » (tourne à gauche) « droite » (tourne à droite) « TDroit » (va tout droit) ou « ½-tour » (demi-tour, reviens en arrière) est choisi.
- « base » consiste à suivre un amer visuel quand plus de cinq boulons ont été détectés.

Chapitre 5.

Briques de base : suivi de couloir.

Le chapitre trois a permis d'introduire la notion de hiérarchie de comportements. Nous avons utilisé cette structure formelle pour présenter la stratégie globale que nous allons utiliser. Celle-ci permet de spécifier chacun des comportements indépendamment les uns des autres. Nous avons segmenté la tâche de retour à la base en une hiérarchie à six niveaux, dont un minimal.
Il s'agit maintenant d'expliciter ces comportements. Ce chapitre va présenter de manière exhaustive le premier de ces comportements, le **suivi de couloir**. Les autres comportements seront présentés plus rapidement dans le chapitre suivant.

5.1. Présentation.

Notre premier comportement est une méthode de navigation locale utilisée fréquemment en robotique en environnement structuré. Nous commencerons donc cette partie par un bref résumé de **l'état de l'art**. Nous présenterons alors très rapidement les **différentes méthodes** que nous avons essayées. Puis nous détaillerons le comportement choisi au final. En plus de représenter l'ensemble du travail expérimental, cette présentation des stratégies testées justifie le minimalisme du comportement finalement choisi. Pour cette raison, les méthodes simples seront détaillées en annexe B.
Trois contraintes ont décidé des choix que nous avons faits dans cette partie :

- Le robot doit bien sur être capable de suivre n'importe quel couloir du bâtiment, murs s'élargissant, avec ou sans obstacle, ou encore murs en verre…
- La trajectoire doit être proche de l'axe médian et stable, pour simplifier le comportement supérieur, « détection d'intersection ».
- Enfin les ressources utilisées doivent être aussi faibles que possible.

5.1.1. État de l'art.

Suivre la direction du couloir est une méthode classique de navigation locale entre deux noeuds d'un graphe topologique. Les graphes sont souvent réalisés pour des navigations en environnements structurés intérieurs. Le plus simple dans ce cas est de représenter un couloir par une arête, et d'apprendre au robot un comportement simple pour suivre un couloir.
Les premiers graphes topologiques sont introduits dans {KUIPERS89}. Le robot suit les murs. Le mur choisi dépend de l'indication donnée dans le noeud du graphe que l'on vient de quitter, comme expliquée en 2.1.2. Dans un couloir, ce comportement le conduit à suivre la bonne direction sans problème. Les auteurs ne précisent pas le comportement adopté quand le robot passe devant une porte ouverte[7].
Cette tâche étant régulièrement utilisée pour la navigation en espace intérieur, d'autres études s'y sont intéressées. Dans {LEBELTEL99}, l'auteur spécifie un programme bayésien très simple permettant de suivre un mur. Il étudie ensuite le comportement adopté par son robot lorsque celui ci se trouve dans un couloir (en particulier quand il en sort). Le robot utilise simplement huit capteurs infrarouges, qui lui renvoient la distance aux murs.
D'autres méthodes utilisent des cameras comme capteurs. Dans {MENG93}, des réseaux de neurones extraient de l'image les coordonnées de la ligne d'intersection entre les murs et le

[7] Le comportement souhaité est sans doute qu'il franchisse la porte puisqu'il exécute une tâche plus générale que suivre un couloir.

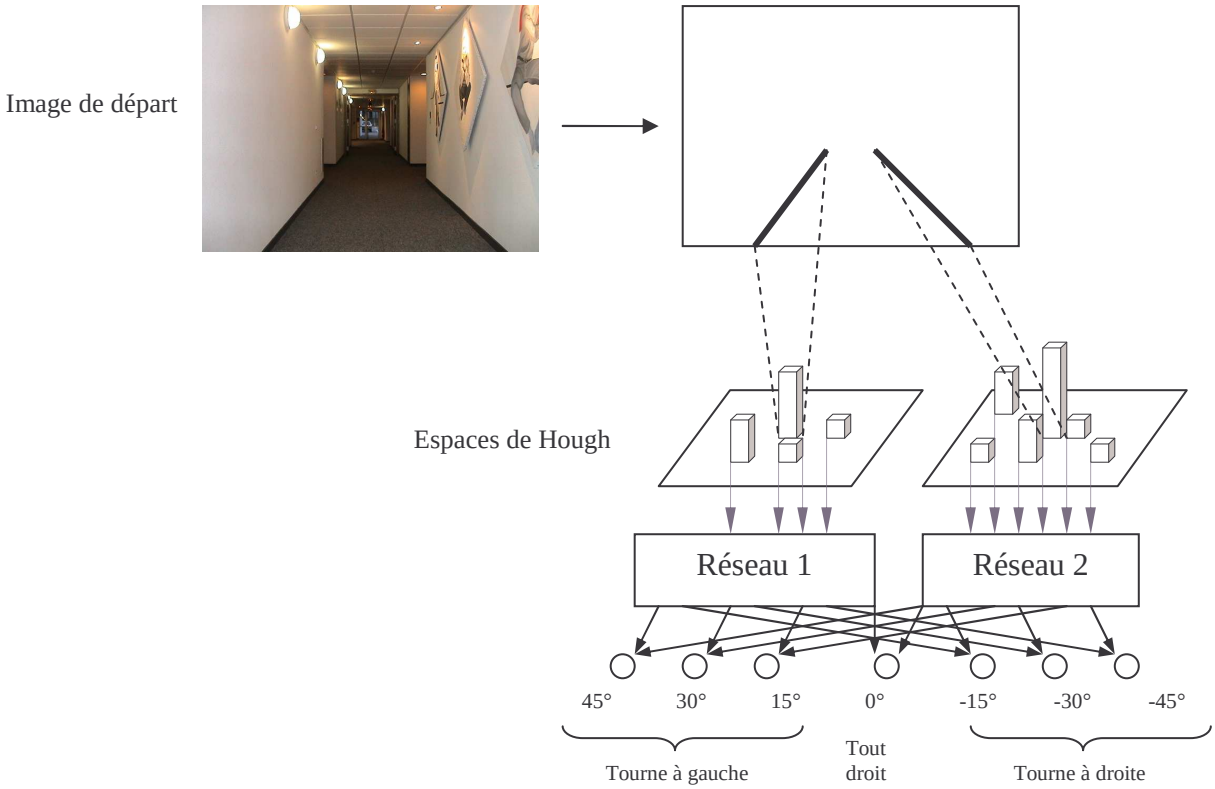


Figure 5-1 : Deux réseaux de neurones suivent séparement l'intersection entre le plancher et le mur de gauche ou le mur de droite. La fusion des deux comportements permet de loger le couloir.
Source: inspiré de {DESOUZA02}, p. 246.

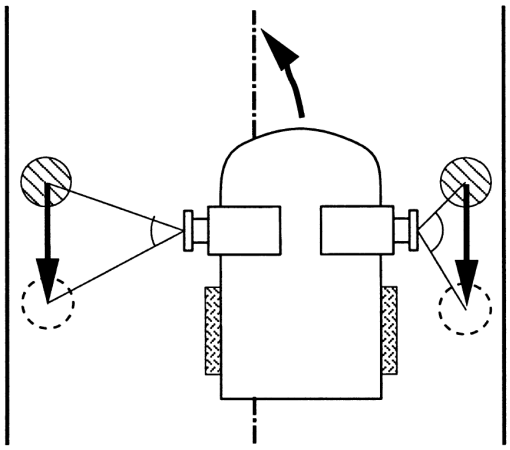


Figure 5-2 : Le robot s'aligne sur le milieu du couloir, en équilibrant les flots optiques détectés à droite et à gauche.
Source: {FRANZ00} p. 139, fig. 3.

plancher. Le premier réseau permet ainsi de suivre le mur gauche, le second le mur droit. Une fusion des deux suivis permet de longer le couloir (cf. Figure 5-1).
Une autre méthode présentée dans {COOMBS93} permet de guider le robot en comparant les flots optiques détectés à droite et à gauche. Deux caméras, une pour chaque coté, estiment la vitesse du fond de l'image. Lorsque la vitesse enregistrée est plus élevé à droite qu'à gauche, le robot tourne à gauche, s'éloignant du mur droit (la vitesse détectée à droite diminue) et se rapprochant du mur de gauche (la vitesse détectée à gauche augmente), comme montré Figure 5-2. Un équilibre est atteint quand le robot est à égale distance des deux murs, sur la ligne en pointillé sur la figure. L'idée de cette méthode est reprise dans {TSAKIRIS00}, en utilisant une caméra panoramique. Une estimation de la vitesse de défilement des murs est calculée en utilisant la géométrie propre à la caméra panoramique. Les auteurs prouvent alors la stabilité du système cinématique ainsi défini.

Au final, on voit que deux méthodes sont généralement utilisées. La première consiste à calculer, directement ou non, la distance aux deux murs, et à placer le robot à égale distance des deux. Dans {LEBELTEL99} ou {KUIPERS89}, des capteurs mesurant la distance à l'obstacle sont utilisés, ce qui donne directement la commande à appliquer. Dans {COOMBS93} et {TSAKIRIS00}, on estime la vitesse de déplacement pour placer le robot en équilibre sur la ligne du milieu.
Une seconde méthode cherche à reconstruire un modèle du couloir, généralement par la vision, puis déplace le robot dans la position voulue sur ce modèle. C'est ce qui est fait dans {MENG93}. D'autres équipes ont proposé des algorithmes reconstruisant un modèle du couloir. Le système FINALE, présenté dans {DESOUZA02}, établit un modèle beaucoup plus complet, intégrant portes et intersection, dans lequel le robot peut se localiser de manière exhaustive.

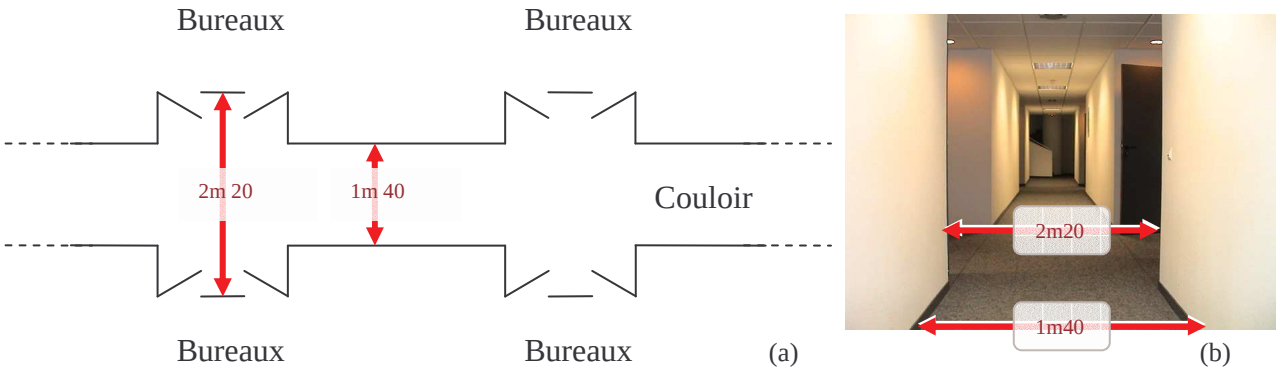


Figure 5-3 : Architecture des couloirs du bâtiment dans lequel se déplace le robot.
(a) Plan sommaire d'un couloir.
(b) Photo du couloir.

5.1.2. Plan du travail effectué.

Pour notre part, le principal problème vient de la forme des couloirs du bâtiment dans lequel le robot se déplace. Les couloirs ne sont pas simplement de longs murs droits et parallèles, ou même gardant toujours le même écartement. Comme montré Figure 5-3, les couloirs s'élargissent tous les 10m, pour former une pièce ouverte de 4m sur 4m, devant chaque porte.

Nous allons proposer différents algorithmes utilisant l'une ou l'autre des méthodes vues dans 4.1.1. Nous n'utiliserons pas les techniques de vision par ordinateur pour cette tâche simple, ce n'est pas le sujet de ce travail. Le but est de placer le robot au milieu du couloir, quelque soit la position adoptée au départ, grâce au capteur-laser. Une formalisation bayésienne de ces programmes sera proposée. Les résultats expérimentaux seront présentés, ainsi qu'une estimation qualitative du comportement obtenu.
Pour chacun de ces algorithmes, nous avons utilisé un évitement d'obstacle simple, comme défini dans {LEBELTEL99}. Cet évitement d'obstacle conduit le robot à suivre la vitesse donnée en consigne si les obstacles détectés sont loins, et à adopter une vitesse lui permettant de contourner l'obstacle quand celui ci est trop proche. Nous le présentons dans un premier paragraphe. Puis nous exposerons en détail les méthodes mises en pratique pour résoudre ce problème. Les méthodes vont en se complexifiant, tout en gardant comme but commun de proposer des comportement réactifs et simples, n'extrayant qu'un minimum de données des capteurs avant de décider quelle action effectuer.

5.2. Evitement d'obstacle simple.

5.2.1. Présentation.

Nous présentons ici un évitement d'obstacle simple. Le robot considère un seul obstacle à la fois, le plus proche. Il est considéré statique, ou quasi-statique. Le seul comportement du robot va être de détourner la trajectoire pour ne pas rentrer en collision avec l'obstacle. Il n'y a pas de superviseur qui garantirait par exemple que la trajectoire choisie garantisse de trouver un chemin pour contourner l'obstacle. Un couloir étant un lieu particulièrement étroit, cela s'avère pourtant nécessaire, comme montré Figure 5-4.

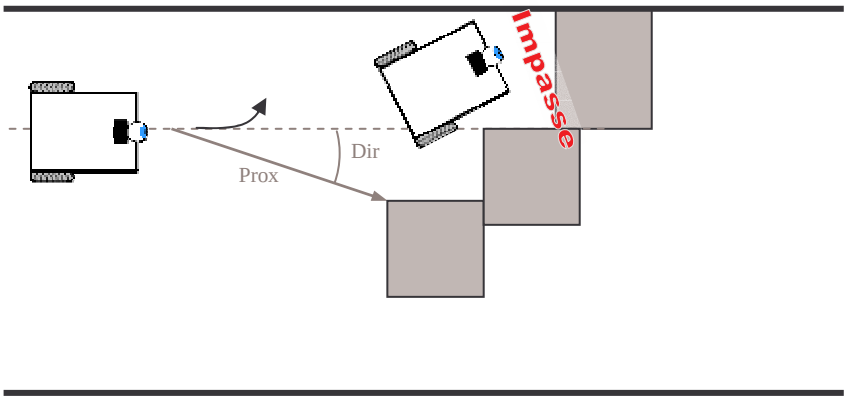


Figure 5-4 : Le robot détourne sa trajectoire de la manière la plus simple possible. Cela peut le conduire à une impasse.

5.2.2. Spécification.

- **Variables:**

Vconsigne: vitesse de consigne, à adopter si aucun obstacle n'est détecté.
Vrot: vitesse corrigée pour éviter l'obstacle.
Dir: direction de l'obstacle le plus proche.
Prox: distance à l'obstacle le plus proche.
H ∈ { evit, consigne }: comportement adopté.

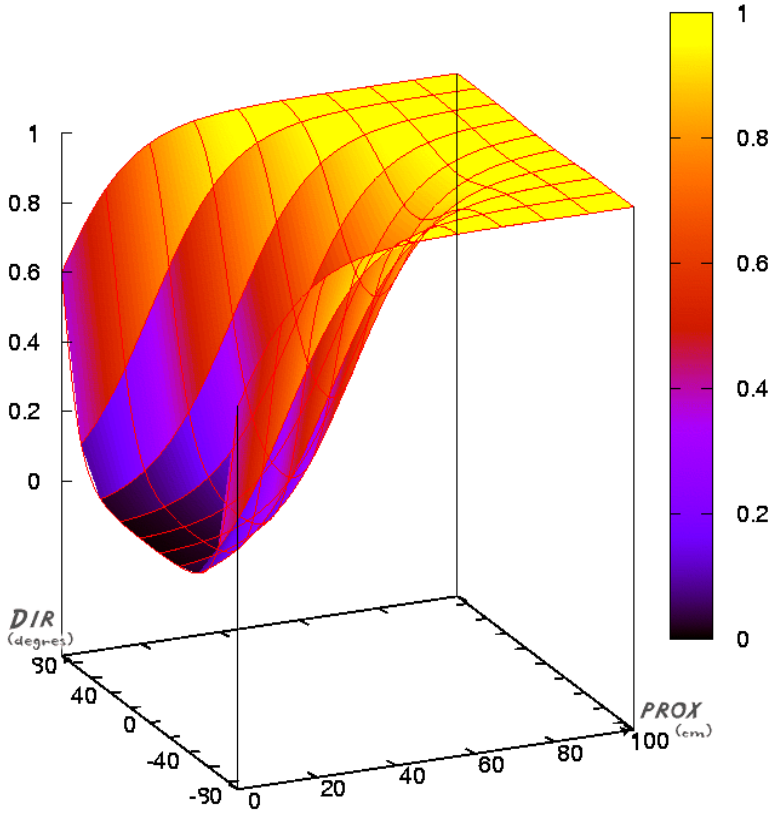


Figure 5-5 : La dépendance P([H = consigne] | Dir, Prox) est spécifié comme fonction des valeurs de Dir et Prox par une sigmoïde. On voit que le robot ne tient aucun compte d'un obstacle situé à plus de 1m de lui.

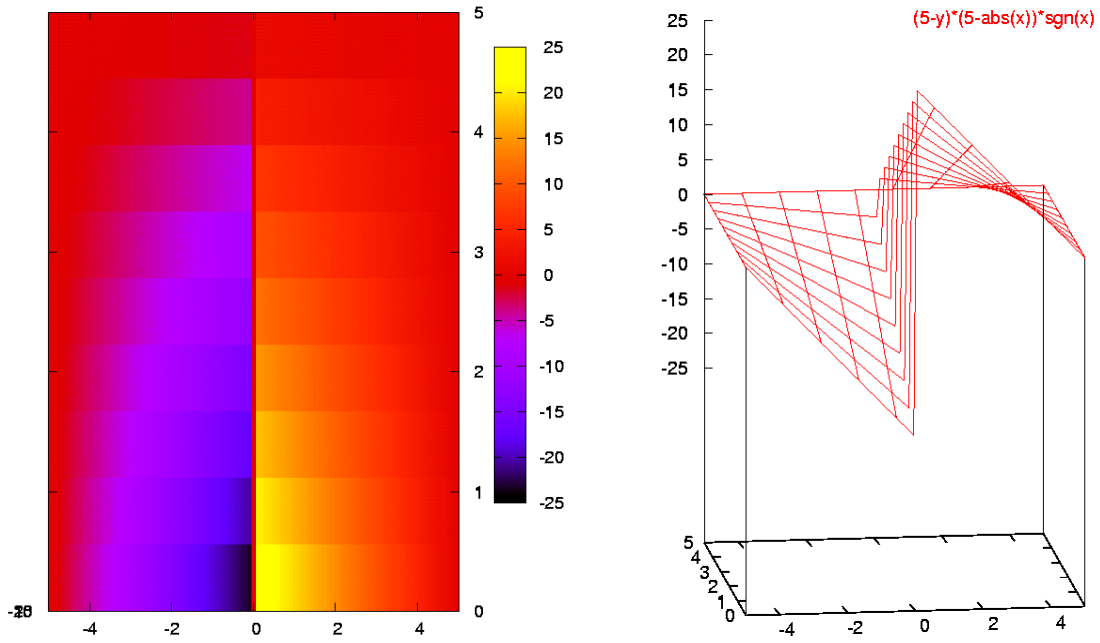


Figure 5-6 : Fonction m(Dir, Prox). Le robot tourne plus vite si l'obstacle est en face de lui et proche. La discontinuité en Dir=0 (obstacle en face),

- **Décomposition:**

$$P(\text{Vconsigne, Vrot, Dir, Prox, H}) = \left| \begin{array}{l} P(\text{Dir}) \times P(\text{Prox}) \times P(\text{Vconsigne}) \\ \times P(\text{H} \mid \text{Dir, Prox}) \\ \times P(\text{Vrot} \mid \text{H, Dir, Prox, Vconsigne}) \end{array} \right.$$

- **Forme paramétrique:**

P(Dir), P(Prox), P(Vconsigne): on ne fixe pas d'a priori sur ces variables. On utilise donc la loi uniforme.
P([H = evit] | Dir, Prox): on utilise une sigmoïde. Le comportement souhaité est le suivant : plus les obstacles sont proches et en face de lui, plus le robot doit être contraint sur sa vitesse. La fonction utilisée est

$$f_1(d, p) = \frac{1}{(1 + \exp(A - B \cdot d^2 - C \cdot p))}$$

Elle est représentée Figure 5-5. On utilise le carré de la valeur de la variable Dir car celle ci peut être négative, et que nous voulons un comportement symétrique.
P([H = consigne] | Dir, Prox) = 1 - $f_1(d, p)$, par normalisation.
P(Vrot | Dir, Prox, Vconsigne, [H = evit]) = $\mathcal{G}$ (μ(Dir, Prox), σ(prox)). La vitesse dépend uniquement de l'obstacle, c'est à dire des variables Dir et Prox.
P(Vrot | Dir, Prox, Vconsigne, [H = consigne]) = $\delta_{\text{Vconsigne}}$.
Quand les obstacles sont suffisamment loin, on applique exactement la consigne.

5.2.3. Identification.

Les paramètres de f1 (d, p) sont choisis par des essais successifs, jusqu'à obtenir le comportement voulu. La fonction obtenue est représentée Figure 5-5 (A=5, B=2, C=1/5).
Il reste à définir la moyenne et la variance de la gaussienne $\mathcal{G}$. La moyenne μ est représentée Figure **5-6**. Elle garantit que le robot s'éloigne de l'obstacle, d'autant plus vite que celui-ci est proche. La variance a été choisie fixe, pour simplifier. On aurait pu prendre une fonction croissante selon la distance (l'incertitude sur la commande à exécuter diminue quand l'obstacle se rapproche).

5.2.4. Question.

On connaît la vitesse de consigne et les paramètres de l'obstacle. La question à poser est donc :

$$P(\text{Vrot} \mid \text{Dir, Prox, Vconsigne}).$$

5.2.5. Evaluation du comportement.

Ce comportement est utilisé régulièrement par les différents robots de l'équipe. Il est suffisant bien pour permettre au robot de trouver une solution dans des cas très simples. Si un humain par exemple se trouve devant son passage, le robot fera un petit écart pour l'éviter, puis reprend sa trajectoire initiale. C'est tout ce qu'on voulait.
On a implémenté le même algorithme pour le calcul de Vtrans, la vitesse de translation du robot, en changeant les paramètres. En effet, dans un couloir assez étroit, le robot ne dispose pas d'une grande marge pour s'écarter de son chemin, il est préférable qu'il ralentisse en premier lieu en présence d'un obstacle, et qu'il s'écarte ensuite seulement, si l'obstacle ne s'est pas dégagé.

Le principal intérêt de comportement est d'offrir **un moyen simple de tester la stabilité des systèmes** que nous présenterons dans la suite. En effet, ce n'est pas tout de savoir rester au milieu du couloir, il faut aussi pouvoir y retourner si on s'en écarte. Pour tester cette robustesse, le plus facile est de poser un carton au milieu du passage. Le robot s'écarte alors de sa trajectoire pour l'éviter. On peut alors voir s'il est capable de la retrouver.
Nous allons maintenant présenter les comportements de suivi de couloir. Ils se présentent tous de la même façon : on a trois ensembles de variables :

- les variables capteurs, connues (Capt).
- les variables intermédiaires, inconnues (Unknown).
- et la vitesse de rotation recherché, Vconsigne.

On posera alors la question P (Vconsigne | Capt). En pratique, on ajoutera les variables d'évitement d'obstacle, et on posera la question :

P (Vrot | Capt, Dir, Prox).

5.3. Quatre stratégies pour suivre un couloir...

Comme expliqué au début du 4.1.2, suivre un des murs ne suffit pas, car les couloirs du bâtiment dans lequel évolue le robot ne sont pas uniformément droit. Nous allons donc chercher une stratégie minimaliste permettant de réaliser cette tâche. Pour cela nous avons tout d'abord tester des programmes utilisant directement les capteurs en une loi de commande : odométrie, valeur d'une mesures-laser à droite et à gauche, ou angle de la mesure la plus élevé. Il s'est avéré après expérimentation que ces stratégies n'étaient pas suffisante. Nous avons alors implémenté un algorithme extrayant un modèle simple du couloir : distances aux murs et angle de l'axe du couloir.

5.3.1. Les quatre stratégies proposées.

Nous avons appliqué quatre stratégies différentes pour le robot. Celles ci sont présentées de la plus simple à la plus complexe. Ces stratégies sont toutes basées sur la même idée. Il s'agit de calculer Vrot, la vitesse de rotation à appliquer en fonction de la lecture capteur. Nous utilisons essentiellement le capteur laser, qui renvoie 360 mesures à chaque instant. Ce grand nombre de valeurs nécessite impérativement un prétraitement pour être utilisé. C'est dans cette partie que vont se trouver les différences entre les méthodes présentées. Chacune utilise une partie différente des informations contenues dans les mesures pour diriger le robot.

- La première méthode utilise simplement les paramètres renvoyés par l'odométrie. On approxime l'axe médian par la droite (Y = 0) du repère odométrique. La loi de commande alors se déduit simplement. Evidemment, le résultat obtenu est mauvais. Le cumul des erreurs rend l'approximation par la droite (Y = 0) vraiment trop grossière. La trajectoire diverge jusqu'à ne plus suivre du tout le couloir.
- La seconde méthode cherche à positionner le robot par rapport aux murs du couloir. Le robot utilise directement les valeurs des mesures laser correspondant aux 90°, et -90° (à gauche et à droite, l'angle 0° correspondant à l'axe du robot). On intègre à la loi de commande les mesures du temps précédent pour stabiliser la trajectoire.
- La troisième méthode oriente le robot dans la direction du couloir (ce que ferait un humain). Les résultats en stabilité et en direction sont excellents. Par contre, le robot ne se place pas correctement au centre du couloir, c'est-à-dire à égale distance des murs. Ce comportement est malgré tout suffisamment bon pour qu'on ait jugé intéressant de l'intégrant dans la hiérarchie, juste en dessous du suivi de couloir. Nous en reparlerons dans la partie 6.4.

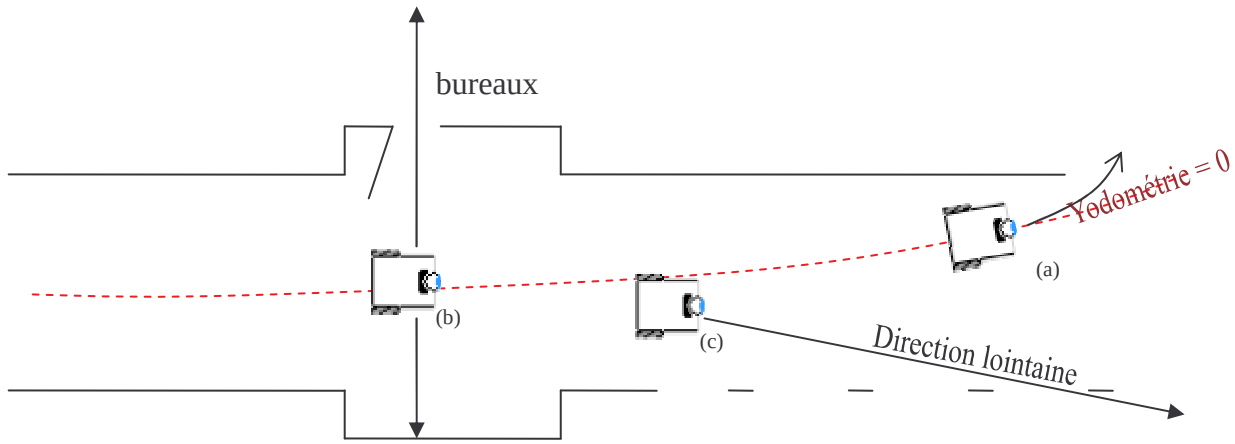


Figure 5-7 : Principales causes de divergence des trois premières méthodes.
(a) A cause du cumul d'erreurs de l'odométrie, la trajectoire du robot diverge.
(b) La seconde méthode est très sensible aux variations dans la distance aux murs.
(c) Le long d'un mur vitré, le suivi de direction lointaine diverge souvent jusqu'à rentrer en collision avec le mur.

- La **quatrième méthode** reconstruit un modèle très simple du couloir pour repositionner le robot au mieux.

Nous ne détaillons pas les trois premières méthodes ici. Elles sont très intuitives et nous pensons que les rapides explications que nous venons d'en donner suffisent. Le lecteur désirant plus de détails sur notre implémentation et les tests effectués est invité à consulter l'annexe B.
Nous nous attarderons par contre plus longtemps sur la dernière méthode, et sur les pré-traitements qu'elle nécessite.

5.3.2. Comportement requis.

Le but de ce comportement est d'obtenir une trajectoire stable, sans oscillation, et qui garantisse que le robot aboutira aux intersections selon un comportement le plus prévisible possible. En effet, pour faciliter la tâche du comportement « détection d'intersection », il est souhaitable que le robot soit toujours orienté dans la direction du couloir, et se trouve à égale distance des deux murs. C'est dans cette position qu'une intersection est la plus repérable à l'aide du capteur laser.
Le robot doit donc suivre la ligne correspondant au milieu du couloir. Dans notre bâtiment, cette ligne est une droite, toujours continue. Elle correspond exactement à la trajectoire que l'on veut faire adopter au robot.

5.3.3. Suite de l'exposé.

Nous allons présenter maintenant la stratégie utilisée dans la hiérarchie. Nous rappelons les deux enjeux de cette présentation. Il s'agit bien entendu tout d'abord d'**exposer le programme bayésien** utilisé. Mais il nous faut aussi **relier ce travail au reste de la hiérarchie**. Cela se fait en définissant les valeurs de crédibilité (C) et d'intérêt (I) de ce comportement.

5.4. En construisant un modèle simple de couloir.

5.4.1. Présentation.

Les comportements précédents ont chacun leurs forces et leurs faiblesses. Suivre la direction du couloir apporte une certaine stabilité à la trajectoire. Mais le prétraitement appliqué ne suffit

pas à connaître de manière sûre la direction à suivre. Il faudrait pour cela fusionner les informations extraites des données laser avec celles venant d'un autre capteur. On pourrait par exemple utiliser la caméra, comme exposé dans {MARTIN02}. Nous ne nous sommes pas orientés vers cette solution car il nous a semblé que cela ne corrigerait pas l'autre défaut de cette méthode : le robot ne se place pas au milieu du couloir, mais à plus ou moins 20cm de l'axe central. En effet, la contrainte en position par rapport aux murs ('Y') est beaucoup moins forte que celle en orientation ('θ').
Par contre, le suivi de mur, même s'il aboutissait sûr un comportement plus instable, plus sensible aux obstacles, ou aux portes ouvertes, et moins rapide à retrouver l'axe après s'en être éloigné, garantissait par contre qu'une fois stabilisé, le robot suive la ligne du milieu de couloir.
Nous allons donc chercher dans cette partie à localiser de manière plus robuste au bruit les murs à droite et à gauche du robot, et l'angle que ces droites forment avec l'orientation du robot. Nous en déduirons alors la décision à prendre sur Vrot. Pour cela, nous ne pouvons plus utiliser directement les mesures capteurs, comme nous l'avions fait plus haut. Nous allons chercher à reconstruire un modèle simple du couloir, contenant uniquement les informations dont nous avons besoin. Pour cela, nous allons utiliser une transformée de Hough sur les 361 mesures données par le capteur-laser.

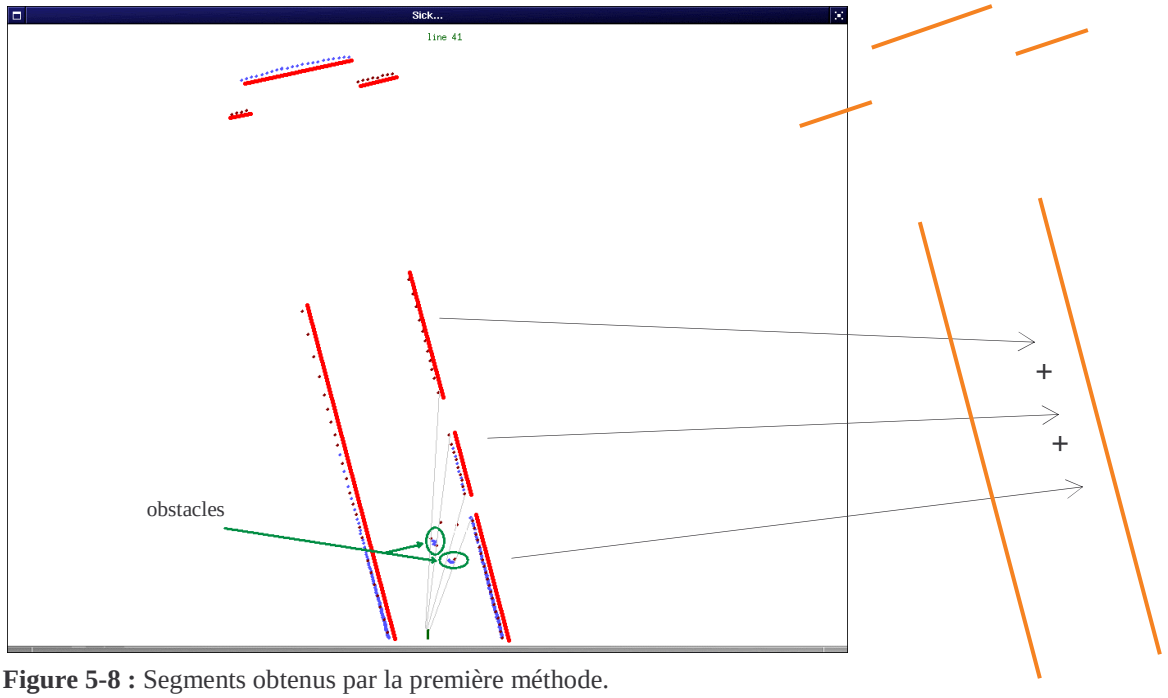


Figure 5-8 : Segments obtenus par la première méthode.
Un mur occulté peut être divisé en plusieurs segments qu'il faudra ensuite rassembler.

5.4.2. Prétraitement des données.

Nous ne désirons extraire des mesures que trois informations : la distance du robot jusqu'au mur gauche d'une part, au mur droit d'autre part, et l'angle que le robot forme avec la trajectoire actuelle du robot. Une première méthode serait d'extraire les segments de l'image formée par les mesures du capteur laser. Pour cela, nous avons tout d'abord appliqué une solution proposée dans {LORIEUX03}. Un premier traitement permet de regrouper les points par voisinage. Puis, on extrait les segments les plus importants de chacun de ces groupes de points. Cela pose plusieurs problèmes. Tout d'abord, on s'attend à obtenir deux segments très importants, représentant les deux murs. Ce n'est pas forcément le cas. Par exemple, le mur peut être occulté en plusieurs endroits par des obstacles, auquel cas cette méthode renverra plusieurs

segments distincts qu'il faudra ensuite mettre en correspondance (cf. Figure 5-8). De plus, de nombreux autres segments sont obtenus, qui ne nous intéressent pas. Au final, on obtient bien tous les segments que l'on désirait, plus d'autres qui nécessiteront un autre traitement avant d'être utilisé pour guider le robot. On préfèrerait une méthode qui nous donne directement les deux segments, si possible est réutilisant les connaissances du temps précédent.
Une autre méthode est proposée dans {VICTORINO02}, pour extraire des segments d'une image de relevés laser. L'auteur propose d'utiliser une transformer de Hough, en se servant des droites trouvées au temps précédent pour intégrer l'erreur sur les mesures laser.

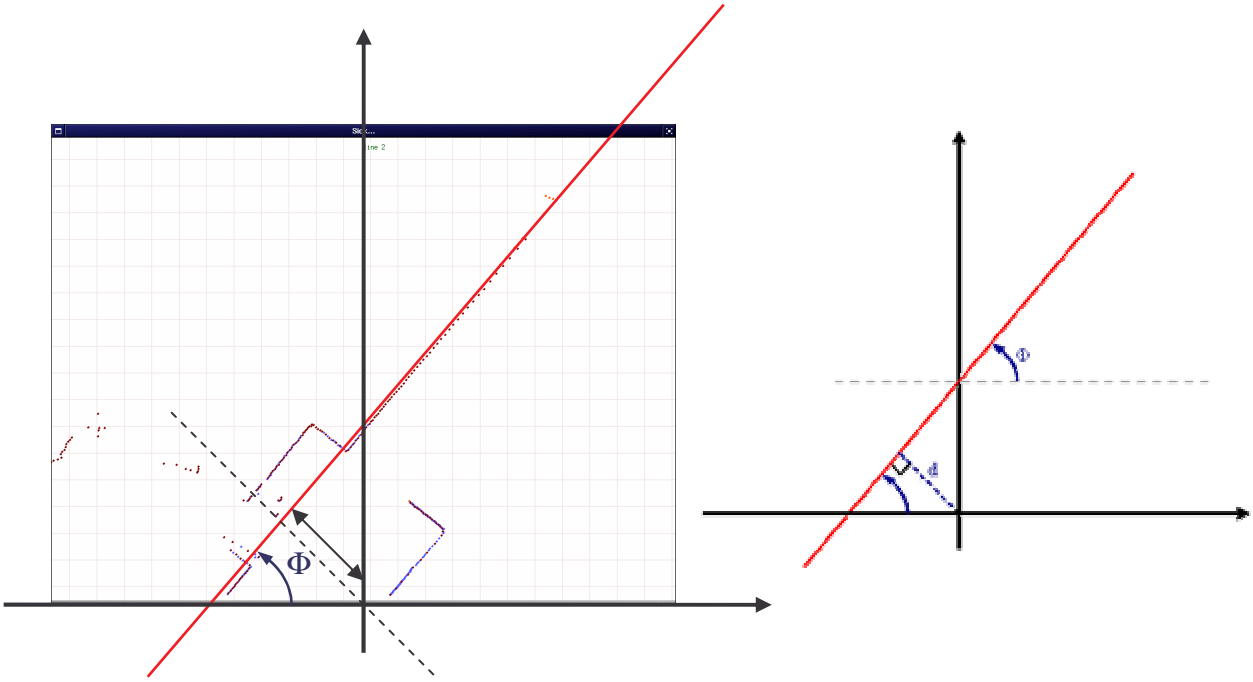


Figure 5-9 : Représentation polaire des droites : on note Φ l'angle de la droite, et d sa distance à zéro. d est toujours positif ou nul.

a) Transformée de Hough.

Pour notre part, nous cherchons à déterminer quatre valeurs :

- **mur_g et mur_d: les distances aux deux murs.**
- **$\theta_{couloir}$: l'angle entre l'axe du couloir et la direction du robot.**
- **certitude: une valeur de confiance sur les mesures que l'on a extraites.**

Cette dernière valeur correspondra à la crédibilité du comportement utilisée dans la hiérarchie de comportements. Elle correspond en fait au nombre de point ayant votés pour la droite lors de la transformée de Hough, ramené dans [0, 1]. Plus cette valeur est élevé, plus le modèle calculé paraît correspondre à la réalité. A l'inverse, plus elle est faible, plus le modèle est douteux.
Nous allons calculer ces valeurs à l'aide d'un vote dans l'ensemble des droites possibles. On représente une droite par son angle d'inclinaison, et sa distance à l'origine, comme présenté Figure 5-9. On la note :

$$(\Phi, d) \in [0, +\infty[\times]-\Pi, \Pi].$$

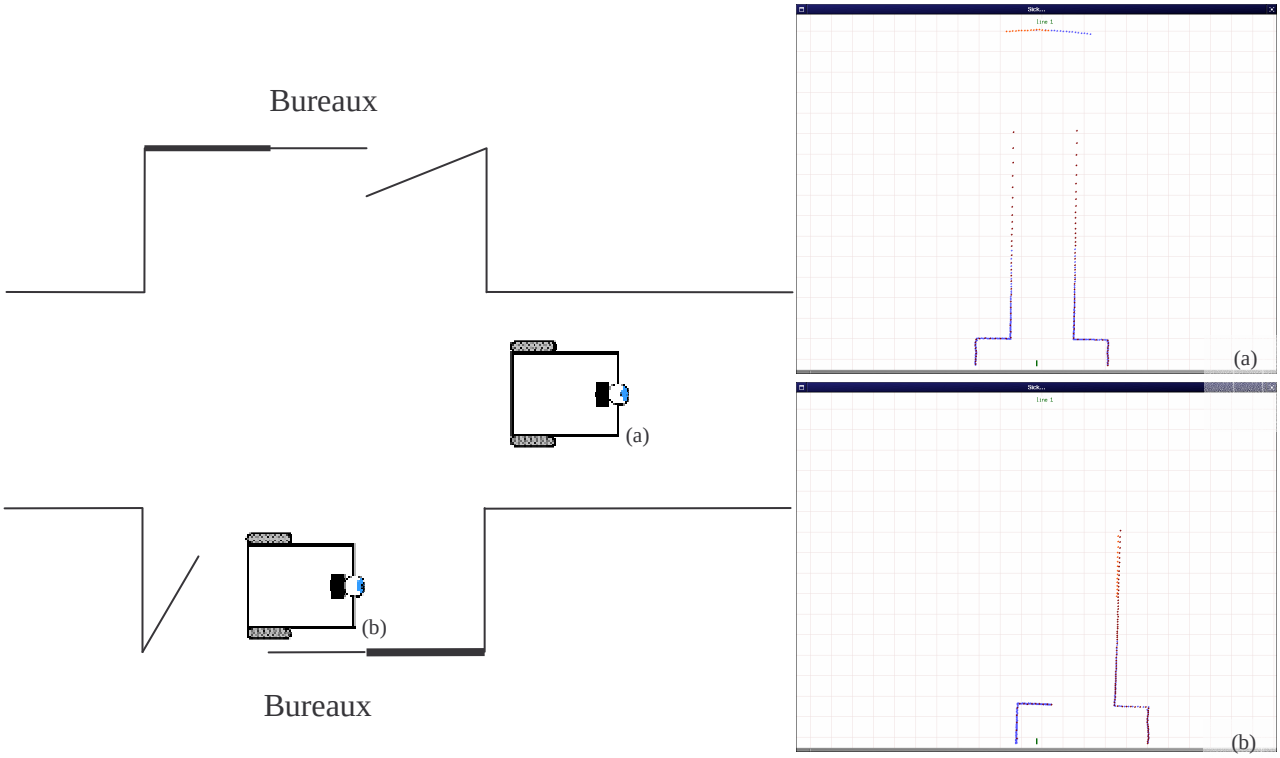


Figure 5-10 : Différentes position du robot par rapport au couloir, et les mesures que renvoie le Sick.
(a) Le robot se trouve entre les deux murs.
(b) Le robot se trouve à l'extérieur du couloir, à droite du mur droit. Ce cas est possible, par exemple quand le robot sort d'un bureau.

b) Deux espaces de vote.

On peut diviser l'ensemble des droites **en deux sous ensembles disjoints** (ce n'est pas une partition, il manque les droites à distance d=0. On ne les prend pas en compte, car elles correspondent à un obstacle en contact) :

$$\Delta_{gauche} = \{ (\Phi, d) \quad tq \quad \Phi \in]0, \Pi[\quad et \quad d \in]0, +\infty[\}$$

$$\Delta_{droite} = \{ (\Phi, d) \quad tq \quad \Phi \in]-\Pi, 0[\quad et \quad d \in]0, +\infty[\}$$

Quelque soit la position du robot par rapport au couloir, le **mur de droite correspondra à une droite de Δ_{droite}.** En effet, comme montré Figure 5-10, le robot ne se trouve pas obligatoirement entre les deux droites qu'il doit détecter comme étant les murs. Dans les parties ou le couloir s'élargit, on considère qu'il doit détecter le mur correspondant à la partie la plus étroite. Mais s'il se trouve à l'extérieur du mur droit par rapport au couloir (position (b) sur la Figure 5-10), le capteur laser ne renvoie aucun point appartenant à ce mur. Le robot ne pourra donc pas le détecter. Au final, soit l'algorithme va conduire à voter pour le mur de droite, dans ce cas, il correspond à une droite de Δ_{droite}, soit il ne votera pas pour une droite correspondant au mur. On vote donc dans deux espaces de droites distincts, Δ_{droite} et Δ_{gauche}.

c) Algorithme.

Comme pour la transformée de Hough classique, chaque point de l'espace de départ (les mesures laser) votent pour toutes les droites possibles qui passe par ce point. Mais dans notre cas, tous les votes non pas la même valeur. Un point vote pour une droite selon un coefficient qui est d'autant plus fort que cette droite est proche de la valeur prédite à l'aide du mur détecté au temps précédent et de l'indice de certitude de cette détection. A l'initialisation de

l'algorithme, la certitude vaut 0. Les autres valeurs n'ont pas d'importance, le calcul ne leur accordant aucune confiance.

Données initiales : mur_d, mur_g, $\theta_{couloir}$, certitude, ensemble des points P détectés.

On note σ = (1-certitude)

On note $f_{droit}(\Phi, d) = \frac{1}{\sqrt{2 \cdot \Pi} \cdot \sigma} \exp\left(\frac{-((d - mur_d)^2 + (\Phi - \theta_{couloir})^2)}{2 \cdot \sigma^2}\right)$

On note $f_{gauche}(\Phi, d) = \frac{1}{\sqrt{2 \cdot \Pi} \cdot \sigma} \exp\left(\frac{-((d - mur_g)^2 + (\Phi - \theta_{couloir})^2)}{2 \cdot \sigma^2}\right)$

Initialiser l'histogramme $VOTE_{droit}$ à 0.
Initialiser l'histogramme $VOTE_{gauche}$ à 0.

Pour tout point P détécté par le laser:
  Pour toute droite (Φ, d) tq $P \in (\Phi, d)$:
    Si $(\Phi \in]-\Pi, 0[)$ $VOTE_{droit}[\Phi, d]$ += $f_{droit}(\Phi, d)$
    Si $(\Phi \in]0, \Pi[)$ $VOTE_{droit}[\Phi, d]$ += $f_{gauche}(\Phi, d)$

$(\theta_{couloir}, mur_g, mur_d) = \underset{\theta, d, g}{\operatorname{argmax}} (VOTE_{droit}[\theta, d] + VOTE_{gauche}[\theta, g])$

certitude = $\max_{\theta, d, g} (VOTE_{droit}[\theta, d] + VOTE_{gauche}[\theta, g])$

Résultat final : mur_d, mur_g, $\theta_{couloir}$, certitude

d) Evolution de l'espace de Hough au cours du temps.

La Figure 5-11 présente un **exemple simple d'exécution**. Le robot est placé dans un couloir, et il ne bouge pas. On cherche simplement à voir de quelle manière le couloir est détecté, et vers quelles valeurs la méthode converge. Au premier passage dans l'algorithme, la certitude est nulle. La variance est donc importante. Tous les votes sont équivalents. On obtient un premier modèle du couloir, très incertain. Au temps suivant, la certitude va augmenter si la transformée de Hough détecte les mêmes droites que au cycle 0. On obtient les mêmes résultats pour $\theta_{couloir}$, mur_g et mur_d. Par contre, la certitude a augmentée. En effet, pour les droites détectées, la valeur des votes est plus forte. On converge rapidement vers une certitude élevée (proche de 1) pour les deux droites détectées lors du premier cycle de calcul.

Quand le robot se déplace, cette détection est appliquée avec des images différentes du couloir à chaque passage. Il faut donc définir un modèle de prédiction des trois valeurs $\theta_{couloir}$, mur_g et mur_d en fonction des vitesses Vrot et Vtrans appliquées au robot. On va simplement chercher $(\theta(t), X(t), Y(t))$ connaissant $\theta(0)$, $X(0)$, $Y(0)$, Vrot et Vtrans. On considère que Vrot et Vtrans restent constantes. En particulier, on ne considère pas de phase d'accélération. Cette hypothèse semble appropriée car les différences de vitesses demandées au robot sont généralement faibles, surtout par rapport aux capacités d'accélération de ses moteurs. Le système dynamique est représenté Figure 5-12. On utilise le repère déjà utilisé précédemment: X correspond à la longueur du couloir, Y à la largeur, et θ est nul quand le robot suit la direction du couloir.

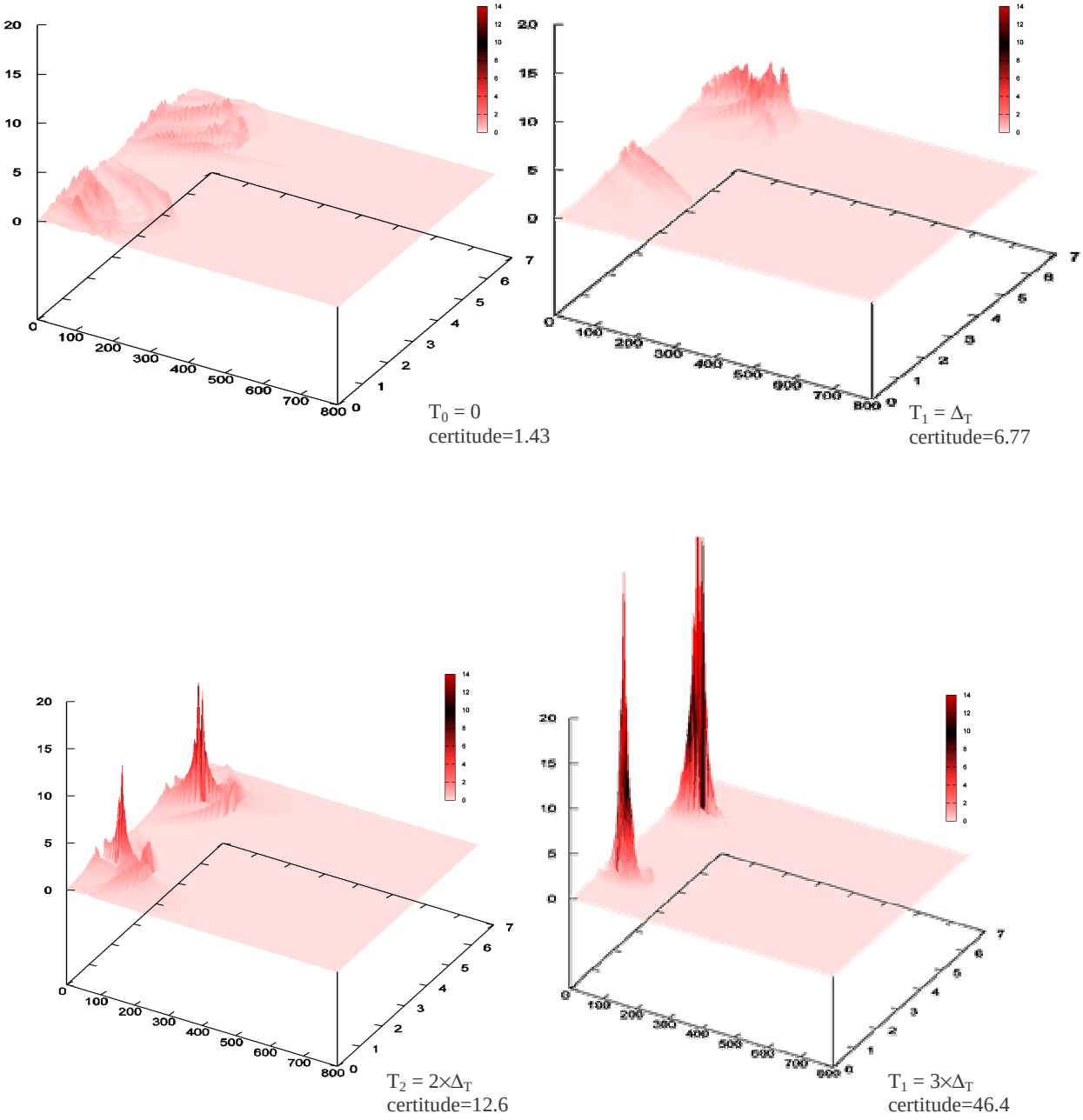


Figure 5-11 : Augmentation de la certitude lors d'une série de transformée de Hough sur une même image de données laser.

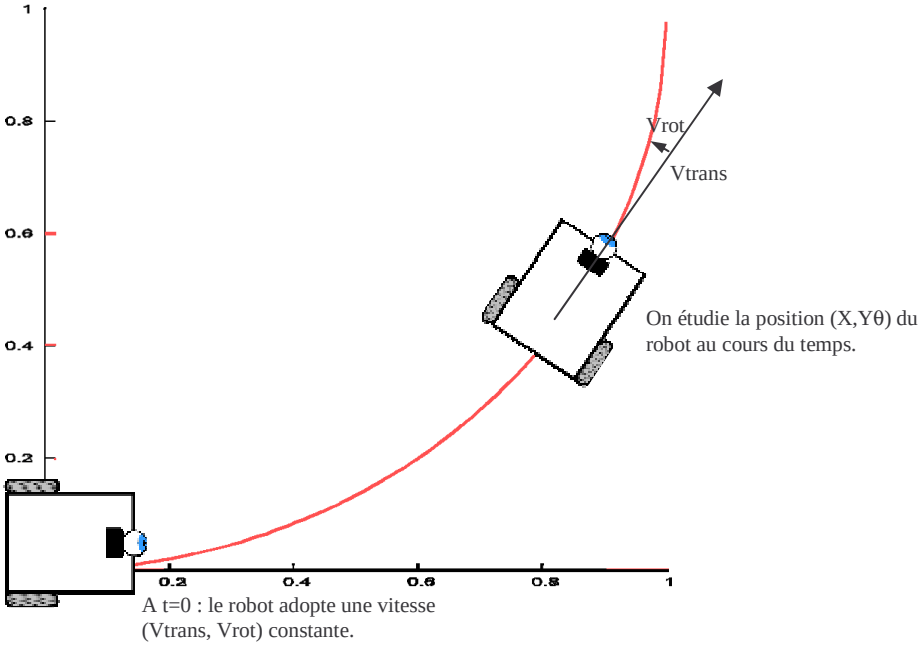


Figure 5-12 : Calcul du déplacement du robot entre deux calculs de la transformée de Hough. Le robot suit une trajectoire de la forme (sin(t), 1-cos(t)).

On obtient le système suivant :

$$\begin{cases} \vec{V} = \text{Vtrans} \cdot \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \\ \theta(t) = \theta_0 + Vrot \cdot t \end{cases}$$

D'où:

$$\begin{cases} \dot{X}(t) = Vtrans \cdot \cos(\theta_0 + Vrot \cdot t) \\ \dot{Y}(t) = Vtrans \cdot \sin(\theta_0 + Vrot \cdot t) \end{cases}$$

$$\begin{cases} X(t) = X_0 + \dfrac{Vtrans}{Vrot} \cdot \sin(\theta_0 + Vrot \cdot t) \\ Y(t) = Y_0 + \dfrac{Vtrans}{Vrot} \cdot (1 - \cos(\theta_0 + Vrot \cdot t)) \end{cases}$$

On obtient finalement:

$$\begin{cases} mur_g^{prédit} = mur_g - \dfrac{Vtrans}{Vrot}(1 - \cos(\theta_0 + Vrot \cdot t)) \\ mur_d^{prédit} = mur_d + \dfrac{Vtrans}{Vrot}(1 - \cos(\theta_0 + Vrot \cdot t)) \\ \theta_{couloir}^{prédit} = \theta_{couloir} + Vrot \cdot t \end{cases}$$

Pour intégrer l'incertitude sur le mouvement au modèle, il nous suffit de diminuer la valeur de la certitude obtenue au temps précédent. On choisit de la diminuer d'autant plus que la vitesse est élevée:

$$\text{certitude} = \text{certitude} - K \cdot (|Vrot| + |Vtrans|), \qquad K > 0.$$

Il ne nous reste plus désormais qu'à utiliser les valeurs calculées pour diriger le robot. Nous explicitons le programme comme plus haut à l'aide d'une formalisation bayésienne.

5.4.3. Programme bayésien.

a) Loi de commande.

On peut formaliser le comportement adopté en fonction de ce modèle par une loi de commande :

$$\begin{cases} mur_g = mur_d & (1) \\ \theta_{couloir} = 0 & (2) \end{cases}$$

b) Spécification.

- **Variables:**

Vrot: vitesse de rotation du robot.
$\theta_{couloir}$, mur_g, mur_d: valeurs renvoyées par la transformée de Hough.

- **Décomposition:**

$P(Vrot, \theta_{couloir}, mur_g, mur_d) = P(\theta_{couloir}) \times P(mur_g) \times P(mur_d) \times P(Vrot \mid mur_g, mur_d, \theta_{couloir})$.

- **Forme paramétrique:**

$P(\theta_{couloir})$, $P(mur_g)$, $P(mur_d)$: nous n'avons pas d'*a priori* sur ces variables. Nous choisissons une loi uniforme.

$$P(Vrot \mid mur_g, mur_d, \theta_{couloir}) = \zeta \begin{pmatrix} \omega(mur_g, mur_d) + \omega(\theta_{couloir}), \\ \sigma(mur_g, mur_d) + \sigma(\theta_{couloir}) \end{pmatrix}$$

Comme précédemment, la vitesse de rotation est une gaussienne centrée autour d'une moyenne entre deux comportements, garantissant chacun le respect d'une des lois de commandes.

c) Identification.

- On choisit de fixer les variances à 1, pour simplifier. Il nous reste maintenant à spécifier les vitesses de rotation ω.

$$\omega(mur_g, mur_d) = K_1 \times (mur_g - mur_d), \qquad K_1 > 0.$$

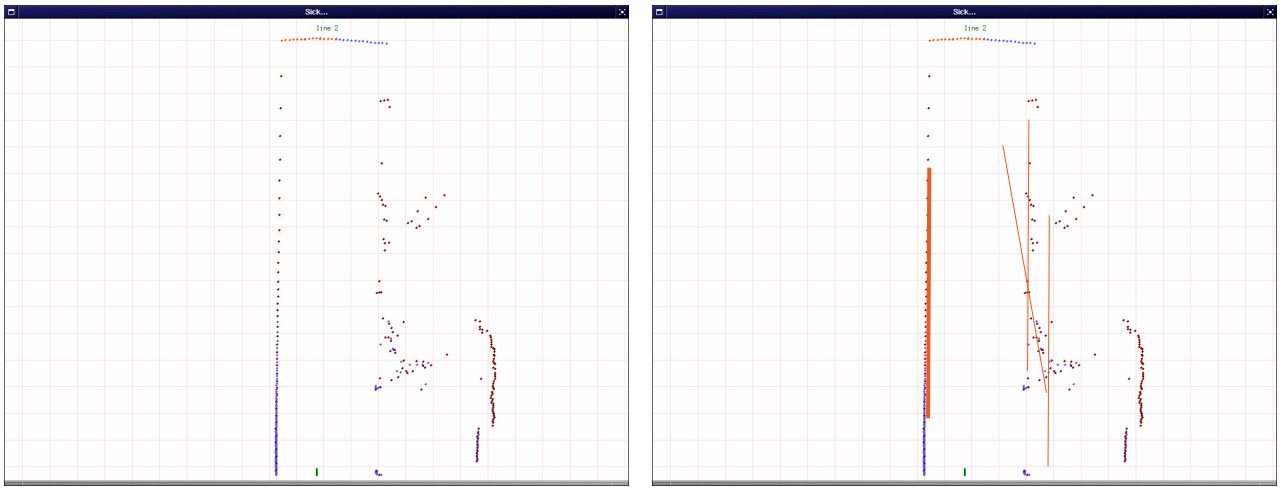
- Quand le robot se rapproche du mur de gauche, il tourne vers la droite ($\omega < 0$).

$$\omega(\theta_{couloir}) = K_2 \times \theta_{couloir}, \qquad K_2 > 0.$$

Si $\theta_{couloir}$ est positif, il faut que le robot tourne vers la gauche.

d) Question.

$$P(Vrot \mid \theta_{couloir}, mur_g, mur_d).$$



(a) (b)

Figure 5-13 : Image du laser obtenue dans un couloir dont un des murs est vitré.
(a) Sans a priori, la transformée renvoie plusieurs droites équiprobables, la plupart ne correspondant absolument pas au mur.
(b) En déduisant de l'autre mur détecté les paramètres du mur vitré, on obtient les paramètres exacts par la transformé de Hough.

5.4.4. Evaluation du comportement.

La **première évaluation** concerne uniquement la loi de commande. On considère donc des cas suffisamment simples pour que la transformée de Hough donne des résultats parfaits. On constate alors que le comportement du robot est excellent. La trajectoire est stable, le robot retourne rapidement au milieu du couloir après un obstacle. Ce n'est pas surprenant, car la loi de commande est très simple.
Nous intéressons donc **maintenant aux cas réels**. La détection des paramètres du couloir n'est pas toujours parfaite. Chaque cas délicat va être présenté plus en détail, ainsi que les solutions que nous lui avons apportées.

a) Un seul mur détecté.

La transformée de Hough n'est pas toujours capable de détecter correctement les deux murs. Si un des deux murs est occulté (comme montré Figure 5-10-b), la détection sur le côté caché renvoie une valeur absurde. La détection est aussi très mauvaise dans un couloir dont un des deux murs est en verre. La Figure 5-13 présente un exemple de mesures obtenues pour un tel couloir, ainsi que les droites les plus probables calculées.
Dans ces deux cas, les valeurs obtenues sont mauvaises, rendant la navigation incertaine. Mais la **certitude obtenue** pour ces valeurs est aussi très faible. On peut donc **utiliser cette valeur comme diagnostic**. Dans ces deux cas, une des valeurs est correcte. On en déduit alors la valeur la plus probable pour la seconde. Elle se déduit facilement, en faisant l'hypothèse que les couloirs font 1m50 de largeur en moyenne. L'angle du couloir n'est donc calculé qu'avec un des histogrammes de vote.
Dans le cas du couloir vitré, ce **diagnostic amène des résultats excellents**. En effet, le robot n'arrive pas à détecter le mur vitré sans a priori. Au premier cycle, le résultat de la transformée de Hough est faux. Au cycle suivant par contre, il utilise les paramètres de l'autre mur pour en déduire les paramètres les plus probables de la vitre. Quand l'*a priori* est suffisamment fort, la transformée de Hough donne directement le bon résultat, avec une certitude satisfaisante.

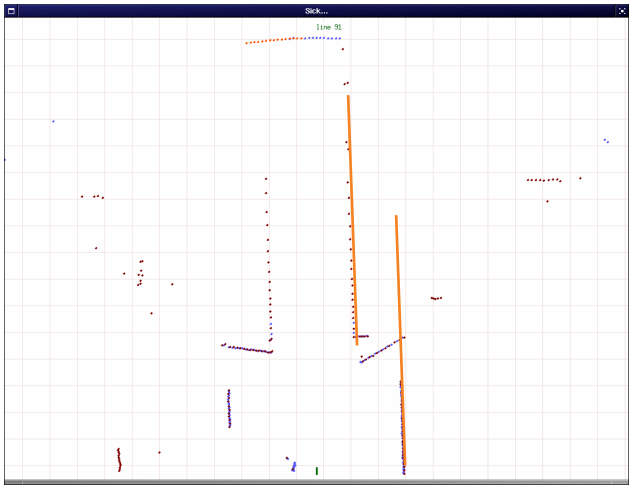


Figure 5-14 : Lorsqu'il se situe à hauteur d'un élargissement, le robot détecte deux droites équiprobables.

b) Obstacles occultant les murs par endroit.

On avait vu en 4.1.8.b. qu'il était difficile de détecter le mur correctement s'il était occulté par endroit par des obstacles (Figure 5-8). La transformée de Hough n'a pas ce problème. Elle détecte une droite, donc cela ne gêne nullement que cette droite soit brisée par endroit. La certitude obtenue est moins grande, mais elle converge toujours. Cela **permet aussi de reconstruire le modèle** du couloir, même si un obstacle est détecté à 90° du robot, alors que le comportement n'utilisant que la distance aux murs à droite et à gauche en était incapable.

c) Fin de couloir.

Le pré traitement cherche à détecter les paramètres des murs, mais il se sert pour cela de la longueur du couloir. En effet, si le couloir se prolonge 10m devant le robot, on aura un grand nombre de points formant une ligne, ce qui augmentera les chances de détecter correctement les murs, ainsi que la certitude obtenue. Ce n'est plus le cas quand le robot arrive en fin de couloir. La transformée a alors beaucoup de mal à détecter correctement les murs. Mais, si les ***a priori*** **sont suffisamment fort, le faible résultat obtenu suffira** à reconstruire un modèle correct du couloir. C'est le cas si le robot a suivi le couloir pendant suffisamment longtemps avant d'arriver à la fin. Expérimentalement, nous avons pu constater que le comportement restait bon si le robot suit le couloir sur 2m au moins avant d'atteindre la fin.

d) Elargissement.

Comme on l'a expliqué plus haut, dans le bâtiment où se déplace le robot, les couloirs s'élargissent à intervalles réguliers. Il arrive donc comme montré Figure 5-14 que la transformée de Hough renvoie deux droites équiprobables, l'une correspondant au mur dans la partie large, l'autre dans la partie étroite. Dans ce cas, on voudrait que la seule droite renvoyée soit celle de la partie étroite, et que la certitude sur cette droite soit augmentée proportionnellement à la certitude sur la seconde droite détectée. Chaque droite probable de l'histogramme de vote ajoute donc une partie de son score à toutes les droites qui lui sont parallèles, et qui sont plus proches du centre qu'elles. En pratique, on extrait les cinq droites les plus probables de l'histogramme, et on les teste deux à deux, renforçant chaque fois l'une au dépend de l'autre.

Nous avons rajouté ici à l'algorithme **un nouvel *a priori* sur la forme des couloirs**. Dans le cas général, cet a priori est justifié. Si le robot doit faire un choix entre deux droites, mieux vaut qu'il ne s'intéresse qu'à celle lui laissant le moins de possibilités. De cette manière, s'il a

fait une erreur, cela le conduira simplement à dévier légèrement de sa trajectoire. Dans le cas contraire, le robot aurait risqué de rentrer en contact avec le mur.

5.4.5. Rattachement à la hiérarchie globale.

Ce comportement est réellement satisfaisant. Le robot parcourt le couloir en restant bien au centre, évite les obstacle puis se replace sur sa trajectoire. Dans les cas pathologiques, il arrive à adopter la trajectoire attendue, malgré certaines hésitations. De plus, nous obtenons pour chaque décision une **valeur permettant de l'évaluer :** la variable de certitude. On a vu qu'elle servait de diagnostic, pour détecter l'absurdité des paramètres d'un des murs, et corriger à l'aide des paramètres calculés pour l'autre mur. Mais elle peut aussi nous servir de diagnostic global, qui nous permettra de changer de comportement si le modèle de couloir construit est trop incertain. Dans ce cas, il nous faudra utiliser un modèle plus simple. Cela correspond à la définition de la valeur de **crédibilité** du comportement. D'un autre côté, si le robot utilise un modèle plus simple pour se déplacer, et que le modèle de couloir extrait par la transformée de Hough paraît correct, le robot doit recommencer à utiliser ce comportement. On utilise la variable de certitude également comme critère d'**intérêt** pour le comportement.

Même si ce comportement est bon, il reste un problème principal: la transformée de Hough extrait un seul modèle de couloir, trois variables avec lequel on effectue tous les calculs suivants. Le robot n'est pas capable de traiter deux modèles équiprobables. Il lui faut faire un choix. On a vu lors des élargissements du couloir que cela nécessitait un a priori supplémentaire pour savoir lequel choisir. Mais le principal problème intervient lors de l'utilisation du modèle choisi au temps précédent pour recalculer le modèle courant. En reprenant le cas simple où le robot affine son modèle sans bouger, on peut arriver à des erreurs importantes. Il suffit en effet que la première transformée de Hough renvoie deux modèles équiprobables et que le robot choisisse le mauvais pour qu'à partir du temps suivant, la transformée de Hough ne renvoie que des droites fausses. La méthode converge alors vers une erreur. On obtient finalement un Dirac sur une valeur fausse, ce qui rend par la suie difficile la correction de cette erreur.
Dans le cas réel, nous n'avons pas réussi à placer le robot dans une telle convergence. Mais une fois qu'on est arrivé à une certitude suffisamment forte, le second modèle, pourtant tout aussi probable au départ, n'est même plus évoqué. Le robot se trompe, et les données suivantes ne pourront pas démentir cette erreur. Pourtant, si on considère l'histogramme obtenu après application de la transformée de Hough comme une distribution de probabilités, il serait facile de prendre tout le temps en compte les deux hypothèses. Il suffirait alors qu'une information vienne démentir l'hypothèse fausse pour qu'immédiatement le robot cesse d'en tenir compte. Nous allons donc reformuler ce comportement en utilisant le même pré-traitement, .

5.5. En construisant un modèle simple de couloir, seconde version.

La transformée de Hough comme une distribution de probabilité.

5.5.1. Présentation.

Ce comportement va juste consister en une reprise du précédent, formalisé de manière à prendre en compte la possibilité que la transformée de Hough renvoie deux droites équiprobables. Cette formalisation est inspirée des filtres bayésiens, présentés dans {JAVBINSKI76}, en particulier des modèles basés sur les grilles de discrétisation. Nous donnons la formalisation, sans entrer dans les détails de l'implémentation. En particulier, on ne

donnera pas les paramètres exactes des lois de probabilités déjà présentées dans le paragraphe précédent. Il s'agit surtout de montrer qu'on peut ici aussi utiliser le formalisme bayésien en réponse aux incertitudes du modèle.

5.5.2. Programme bayésien.

a) Spécification.

- **Variables :**

Vrot : vitesse de rotation du robot.
$\boldsymbol{\theta_{couloir}}$**, mur$_g$, mur$_d$** : paramètres du modèle du couloir. On désignera ce triplet par la variable **couloir$_T$**.
couloir$_{T-1}$: modèle du couloir calculé au temps précédent.
laser$_0$, …, laser$_T$: ensemble des valeurs renvoyées par le capteur-laser aux temps 0 à T.

- **Décomposition :**

$$P(Vrot, couloir, couloir_{T-1}, laser_{0..T}) = \left| \begin{array}{l} P(laser_{0..T-1}) \times P(couloir_{T-1} \mid laser_{0..T-1}) \\ \times P(couloir_T \mid couloir_{T-1}) \times P(laser_T \mid couloir_T) \\ \times P(Vrot \mid couloir_T) \end{array} \right.$$

- **Forme paramétrique:**

P(laser$_{0..T-1}$) est représenté par une distribution uniforme.
P(couloir$_{T-1}$ | laser$_{0..T-1}$) est donné par une table de probabilité, c'est-à-dire l'espace de Hough normalisé.
P(couloir$_T$ | couloir$_{T-1}$) est modélisé par une gausienne centrée sur les paramètres du couloir au temps précédent, modifiés d'après l'évolution du robot, comme montré dans la partie 5.4.2.a). La variance est une fonction des vitesse de rotation et translation du temps précédent, strictement croissante pour chacune des variables.
P(laser$_T$ | couloir$_T$) = $\prod_{P_i \in laser} P(P_i \mid couloir_T) = \prod_{P_i \in couloir} 1 \times \prod_{P_i \notin couloir} P(P_i)$.
P(Vrot |couloir$_T$) : on reprend la même loi qu'au paragraphe 5.4.

b) Identification.

Tous les paramètres ont été identifiés.

c) Question.

La question à poser est :

$$P(Vrot \mid laser_{0..T}) = \int P(Vrot \mid couloir_T) \cdot \int \left| \begin{array}{l} P(couloir_{T-1} \mid laser_{0..T-1}) \\ \times P(couloir_T \mid couloir_{T-1}) \cdot dcouloir_{T-1}\ dcouloir_T \\ \times P(laser_T \mid couloir_T) \end{array} \right.$$

5.5.3. Description et évaluation du comportement.

Le principe est finalement de faire passer d'un temps à l'autre la distribution de probabilité $P(couloir_T \mid laser_{0..T})$ représenté par l'espace de Hough, puis d'intégrer sur tous les modèles de couloir probables la vitesse à adopter. On utilise donc l'espace de Hough du temps T-1 comme coefficients de vote pour la transformée de Hough au temps T. Puis on intègre les modèles probables ainsi obtenus pour calculer la distribution de probabilité de Vrot. On utilise en fait

une intégrale de Monte Carlo : on tire au hasard vingt modèles différents, ce qui nous donne une approximation suffisante de l'intégrale pour notre tâche.
Le comportement n'est finalement pas très différent du précédent. Nous nous sommes contentés finalement de réécrire le filtrage en utilisant l'équation de Chapman-Kolmogorov, qui constitue la première étape de tous les filtres bayésiens. Les résultats sont sensiblement identiques. On note deux améliorations importantes :

- tout d'abord, la détection ne diverge plus vers un modèle erroné, et des mesures laser qui contredisent le modèle ont une action plus important sur la distribution de probabilités, la lisant beaucoup plus rapidement que dans le cas précédent (3 jeux de mesures suffisent en général) ;
- il n'est pas nécessaire d'ajouter un nouvel *a priori* pour traiter les élargissements de couloir. Deux modèles équiprobable conduisent le robot à se stabiliser au centre de la meilleure manière.

On utilise le maximum de la distribution $P(\text{couloir}_T \mid \text{laser}_{0..T})$ comme valeurs de crédibilité et d'intérêt (C et I). Par contre, ces valeurs n'atteignent pas des valeurs aussi importantes que dans le cas précédent.

5.6. Conclusion et lien avec la suite.

Nous avons **défini un comportement** efficace de suivi de couloir. Nous l'avons de plus **connecté au reste de la hiérarchie** en définissant les deux variables C et I. Enfin, ce comportement va nous permettre de détecter plus facilement une intersection, c'est-à-dire la tâche du comportement supérieur, que nous allons voir dans le chapitre suivant.
Le comportement que nous proposons finalement est **issu d'une démarche** que nous nous sommes attaché à faire ressortir. Les comportements intermédiaires, rapidement évoqués et détaillés en annexe, nous ont servi, par leurs insuffisances, à construire une stratégie satisfaisante qui reste légère. Cette démarche 'dichotomique' **prouve d'une certaine manière le minimalisme** de notre méthode.
Le principal critère que nous avons utilisé pour comparer les comportements est celui de l'empirisme. Nous avons **évalué la stabilité et la robustesse des méthodes** en examinant les résultats expérimentaux. Nous avons aussi cherché à prouver la stabilité de la loi de commande linéarisée par le critère de Yapounof. Mais les causes d'instabilités ou d'erreurs proviennent le plus souvent du modèle sur lequel repose la loi de commande bien plus que sur une instabilité mathématique de la commande. C'est typiquement le cas du suivi de couloir utilisant l'odométrie. Ce critère ne semble donc pas utilisable dans notre cas, raison pour laquelle nous ne l'avons pas évoqué plus tôt. Il nous semble que d'autres critères plus rigoureux que ceux que nous avons utilisés seraient profitables.
Des travaux menés dans l'équipe parallèlement aux notres proposent une méthode de comparaison des différents comportements avec un apprentissage du suivi de couloir posé en référence. Ils sont exposés dans {YGUEL03}. Cette méthode utilise le maximum d'entropie pour généraliser l'apprentissage, trouver les paramètres optimaux des programmes bayésiens et trouver le comportement optimal.

Ce chapitre ne définit en fait qu'un des cinq comportements de la hiérarchie. Nous l'avons largement développé. Nous allons présenter plus rapidement les autres comportements dans le chapitre suivant. Notre hiérarchie sera alors complète, et le robot sera prêt à être testé.

Chapitre 6.

Briques de base : autres comportements.

Nous avons présenté un premier comportement de la hiérarchie introduite dans le chapitre 3. Il nous reste maintenant les quatre autres comportements à exposer. Nous l'avons déjà dit, nous allons moins nous étendre sur ces comportements. Notre but est maintenant plus de donner suffisamment d'éléments au lecteur pour une bonne compréhension du principe des comportements, que de détailler les problèmes rencontrés lors de la formalisation ou de l'implémentation.
Le chapitre se divise donc en quatre paragraphes. Le premier présentera le comportement utilisé pour **détecter une intersection** (§ 6.1). Nous présenterons alors le comportement situé en haut de la hiérarchie, « **suivi d'amer visuel** » (§ 6.3) pour finir par les deux comportements inférieurs, présenté dans la partie intitulée « **Je suis perdu** » (§ 6.4). Nous pourrons alors conclure en présentant les résultats expérimentaux de ce travail.
Les comportements seront toujours présentés de la même manière. Nous rappellerons tout d'abord **la situation** du comportement par rapport au **reste de la hiérarchie**, c'est-à-dire les comportements inférieurs et supérieurs, et les articulations entre eux. Puis nous présenterons les **caractéristiques** de l'environnement que nous avons utilisées pour le comportement, et la stratégie appliquée à ces informations. Nous résumerons alors si cela est nécessaire les principales conclusions que nous pouvons tirer de ce que nous aurons présenté.
Chacun des comportements est détaillé dans une des annexes C, D ou E.

6.1. Détection d'intersection.

6.1.1. Rappel du contexte hiérarchique.

Rappelons tout d'abord le contexte de ce comportement dans la hiérarchie définie dans le chapitre 3. Ce comportement est supérieur au suivi de couloir. Il devient intéressant quand on détecte que le modèle du couloir linéaire présenté dans le chapitre 4 n'est plus suffisant, typiquement **quand le robot croise une intersection**. A ce moment là, le robot ne doit plus continuer tout droit, mais choisir entre quatre directions possibles.
La décision est prise **en fonction des boulons rouges présents au sol**. Le robot doit suivre le couloir marqué par le plus de taches rouges. Le choix s'effectue automatiquement par la structure de la hiérarchie : les comportements supérieurs prennent à leur tour le relais en fonction de la direction correspondant au maximum de boulons rouges. Si le nombre de taches compté est supérieur à cinq, c'est le comportement de suivi d'amer qui sera choisi.
Si le robot s'est trompé, et qu'il ne se trouve finalement pas dans une intersection, ou encore qu'aucun boulon ne marque les couloirs, le comportement « détection d'intersection » perd sa crédibilité, et le robot repart en suivi de couloir.

6.1.2. Stratégie utilisée et variables pertinentes.

a) Variable d'intérêt.

La variable d'intérêt doit devenir vraie quand le robot croise une intersection. Comme « détection d'intersection » est le seul comportement supérieur à « suivi de couloir », le réel associé à cette variable n'est pas utile.

On réalise pour cela la fusion de trois informations extraites des données laser :

- la mesure des distances relevées à gauche du robot ;
- la mesure des distances relevées à droite du robot ;
- la présence d'une droite de forte probabilité perpendiculaire à la direction du couloir.

b) Variable de crédibilité.

Nous avons vous dans le chapitre 3 que ce comportement devait se terminer dès qu'il avait commencé. On fixe donc la variable de crédibilité à 0.

c) Stratégie associée.

La seule action du robot est de compter les marques rouges au sol. Il prend une image des couloirs dans les quatre directions et en extrait pour chacun le nombre de marques rouges. Ceci est fait en deux phases. Tout d'abord en calculant l'image de probabilité. Puis en étudiant les parties connexes de l'ensemble obtenu. Ceci n'est pas réalisé par un programme bayésien, mais par un algorithme classique. On note m(i, T) le nombre de marques détectées au temps T dans la direction i ($i \in$ {droite, gauche, tout_droit, demi_tour }).

6.2. Tourne droite, tourne gauche …

Ces quatre comportements sont encore plus simples que le précédent. Ce sont les comportements supérieurs du comportement précédent au sen de la hiérarchie. Une fois que le robot a détecté une intersection, le compte des marques au sol active l'un d'entre eux. Il devient alors le comportement courant. Dès que le robot a fini sa manœuvre, ce comportement perd sa crédibilité. Le comportement qui lui est inférieur, « suivi de couloir » reprend alors la main.

6.3. Suivi d'amer visuel.

6.3.1. Rappel du contexte hiérarchique.

Les précédents comportements étaient nécessaires pour que le robot réalise correctement le retour à la base. Mais ils ne nécessitaient pas à notre avis une étude plus longue. Par contre, ce comportement nous semble plus intéressant. Nous n'avons pas eu le temps de l'étudier en détail. En particulier, nous n'avons pas réussi à le mettre en œuvre dans le temps qui nous était imparti.
Ce comportement devient intéressant quand le nombre de boulons détectés par le comportement « détection d'intersection » est supérieur à cinq. On considère alors que le robot est assez proche de la base pour passer en recherche visuelle. Si la cible n'a pas été trouvée au bout d'un temps fixe, le robot change de comportement et essaye de rejoindre un couloir. Pour cette raison, le comportement inférieur au suivi d'amer est directement le comportement minimal. Cela se traduit simplement par cette consigne : « si je n'ai pas trouver la base au bout de X secondes, je suis perdu. »

6.3.2. Stratégie utilisée et variables pertinentes.

On ne redonne pas ici les variables de crédibilité et d'intérêt, déjà définies dans le chapitre 3.
Le robot cherche donc la cible de manière visuelle. La cible est marquée par une marque rouge distinctive et voyante. Ce comportement a déjà été étudié dans {COUE00}. Une différence importante intervient dans notre cas : la cible n'est pas toujours en vue. En particulier, le robot commence ce comportement sans voir la cible. Il doit donc s'avancer jusqu'à la détecter, et à ce moment la seulement utiliser le suivi d'amer.

Nous proposons donc une fusion entre un comportement de suivi de la direction lointaine, déjà présenté dans le chapitre précédent, et le suivi d'amer présenté dans {COUE00}. Le comportement à adopter est fixé en fonction de la taille de la cible détectée par la caméra. Lorsque celle-ci est hors de vu, le robot suit donc la direction la plus lointaine.

6.3.3. Conclusions.

Nous n'avons donc pas eu le temps de développer ni de mettre en œuvre correctement ce comportement. Il mériterait une plus longue étude. Deux points en particulier nécessite un approfondissement :

- Tout d'abord, le déplacement en suivant la direction la plus lointaine est une solution de facilité qui ne garanti pas que le robot retrouve la base. Lorsque les études qui ont abouties sur ce travail ont commencé, la possibilité d'utiliser un émetteur ARVA (utilisé pour la recherche de personne ensevelies sous une avalanche) à été évoquée. Ce projet, abandonné pour des raisons techniques, pourrait constituer une solution appropriée.
- Ce comportement doit se terminer par l'enclenchement des prises du robot sur son socle de recharge. Cela nécessite une trajectoire très fine dans la dernière partie de l'approche. Nous ne pensons pas que le suivi d'un seul point de couleur suffise à l'exécuter. Une solution serait d'utiliser une seconde tache de couleur pour se recalibrée, ou encore un motif plus complexe, comme les fonctions p-similaires introduites dans {SCHARSTEIN99}. Ici aussi, une étude plus longue aurait été nécessaire.

6.4. « Je suis perdu » : comportement minimal.

6.4.1. Rappel du contexte hiérarchique.

Nous avons tout d'abord défini un comportement minimal, qui se situe dans la hiérarchie en dessous du suivi de couloir. Il consiste à faire **tourner le robot sur place jusqu'à ce qu'il retrouve des paramètres corrects** et puisse reprendre une marche normale.
Dans le chapitre 4, nous avions défini un comportement permettant de suivre simplement et de manière sûre un couloir : le **suivi de la direction lointaine**. Ce comportement ne respectait pas toutes les spécifications voulues, et nous avons du en présenter un autre plus complexe pour cette tâche spécifique. Par contre, ce comportement est beaucoup plus général, et donne au robot un comportement utile dans de nombreuses situations où le suivi de couloir est impossible, comme par exemple pour sortir d'une pièce.

6.4.2. Stratégie utilisée et variables pertinentes.

a) Le comportement minimal.

Les variables de crédibilité et d'intérêt n'ont pas besoin d'être définies. Le comportement est donc défini de manière très simple. La fonction motrice associe une même valeur pour Vrot à tous les vecteurs de l'espace senseur.

b) Suivi de direction lointaine.

Nous utilisons une même définition pour les variables de crédibilité et d'intérêt. Il suffit pour que ce comportement soit utile que la mesure correspondant à la direction la plus lointaine soit suffisamment élevée.
La stratégie associée a pour sa part déjà été présentée dans le chapitre 4.

6.4.3. Conclusion.

Ces deux comportements garantissent que le robot ne se trouvera pas dans une impasse dont il ne saura pas sortir. Une fois arrivé au bout d'un couloir, le comportent suivant la direction lointaine perd sa crédibilité. Le robot tourne alors sur lui-même jusqu'à retrouver le chemin d'où il est venu. Le robot peut alors se remettre en marche. Un exemple plus complet est détaillé dans l'annexe E.

6.5. Conclusion générale sur les briques de bases.

Cette partie **termine la présentation de l'implémentation** que nous proposons pour résoudre le problème de retour à la base en environnement structuré. Une série de comportements minimaux s'enchaînent successivement en respectant les règles définies dans le chapitre 3. Lorsqu'un comportement ne correspond plus aux variables capteurs courantes, un comportement moins spécifique est lancé. Du point de vue du comportement, cela revient à dire « je n'ai plus assez d'informations pour fonctionner correctement. **Passons à un comportement utilisant moins d'information spécifiques** ».
Au contraire, quand un comportement plus spécifique que le comportement courant correspond aux valeurs courantes, le robot adapte sa stratégie pour l'utiliser. Du point de vue du comportement, cela revient à dire « Mes traitements sont mieux adaptés à la situation courante. Tes traitements sont **insuffisants pour traiter ces valeurs capteur** ».

Le robot est donc **complètement programmé** pour rentrer à la base en utilisant les marques qu'il a placées lors du trajet aller. Nous allons maintenant pouvoir **présenter les expérimentations** qui prouvent la validité de notre implémentation. Nous présenterons tout d'abord une exécution typique. Puis nous montrerons que le robot peut réutiliser les marques lors des exécutions supplémentaires. Nous pourrons alors conclure ce travail en nous ramenant à la théorie générale des cartes bayésiennes développées dans {DIARD03}.

Chapitre 7.

Expérimentations et résultats.

Nous avons défini un ensemble de comportements et une hiérarchie pour les ordonner. L'ensemble de ce travail a été spécifié en programmation bayésienne, ce qui nous a permis de l'implémenter directement sur le robot présenté dans la section 2.1.
Les tests que nous présenterons maintenant n'ont pas pour objectifs de valider chacun des comportements pris individuellement. Nous avons déjà présenté ces résultats expérimentaux dans la partie précédente. La plupart des stratégies simples n'avaient d'ailleurs pour buts que de proposer une solution rapide à leurs spécifications. Seul le suivi de couloir a vraiment été étudié expérimentalement avec soin. Pour réaliser une implémentation complète et fiable, il faudra réaliser le même type d'étude sur chacun des autres.
Nous nous plaçons donc dans un environnement ne présentant pas spécialement de problèmes qui placent un comportement particulier en difficulté. Nous montrerons dans un premier temps que **la hiérarchie de contrôle permet de choisir les comportements corrects** pour réaliser la tâche de retour à la base (§ 7.1). Le robot arrive à retourner à la base en se servant des marques posées lors de son dernier passage. Mais les marques sont laissés en place et peuvent aussi être utilisées lors d'un trajet ultérieur. Nous étudierons cette possibilité dans le paragraphe 7.2, essentiellement à l'aide d'un simulateur. Nous évoquerons aussi le comportement adopté par le robot si ce marquage se dégrade. Enfin, nous avons constaté lors des tests à l'aide du simulateur que les marquages successifs conduisent **finalement le robot à trouver le plus court chemin** jusqu'à la base. Nous présenterons les résultats expérimentaux le montrant dans la partie 7.2, et nous ramènerons la hiérarchie de comportements utilisée à un l'algorithme de recherche des plus courts chemin dans un graphe. Nous aurons alors montré que non seulement la hiérarchie de comportements est une structure valide pour réaliser une tâche complexe, mais que, en plus, son utilisation peut conduire à des convergence vers des méthodes plus puissantes que celles envisagées au départ.

7.1. Retour à la base.

Dans un premier temps, nous amenons le robot à la main dans une des ailes du bâtiment. Des carrés de papier rouge sont placés à chaque intersection traversée. Puis l'exécution du programme est lancée. Les images les plus intéressantes de la séquence de retour vers la base sont présentées dans les nombreuses figures de cette partie. Le robot est amené dans un des bureaux d'une des ailes du bâtiment. Il doit retourner jusqu'à la halle robotique, qui est situé au bout du couloir principal de l'INRIA (tout en haut du plan Figure 3-1).
Les figures présentées nous semblent suffisamment explicite pour ne pas nécessiter d'autres commentaires. Nous explicitons, dans l'ordre :

- la sortie du bureau et le recentrage dans l'axe du couloir (Figure 7-1);
- Un évitement d'obstacle simple (Figure 7-2);
- L'arrivée dans une intersection (Figure 7-3);
- Le choix de la direction à prendre (Figure 7-4) ;
- Et enfin l'arrivée dans la halle robotique, aboutissement de la tâche à exécuter (Figure 7-5).

Le robot a bien rempli la tâche demandée. Ces essais ont été répétés de nombreuses fois, pour étudier exactement l'enchaînement des actions. Cette exécution est très représentative du cas de figure général. Elle est conforme à ce que nous attendions après avoir spécifié la hiérarchie de comportement. Ceci constitue une validation par l'expérience de notre architecture.

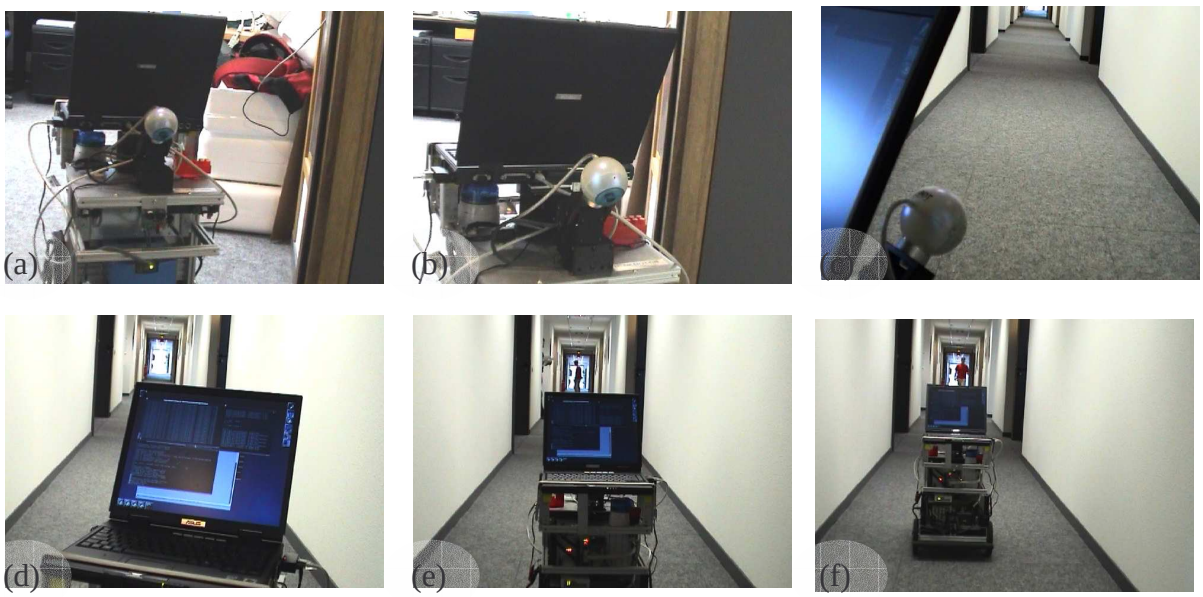


Figure 7-1-a : Le robot sort du bureau et s'orient dans la direction du couloir.

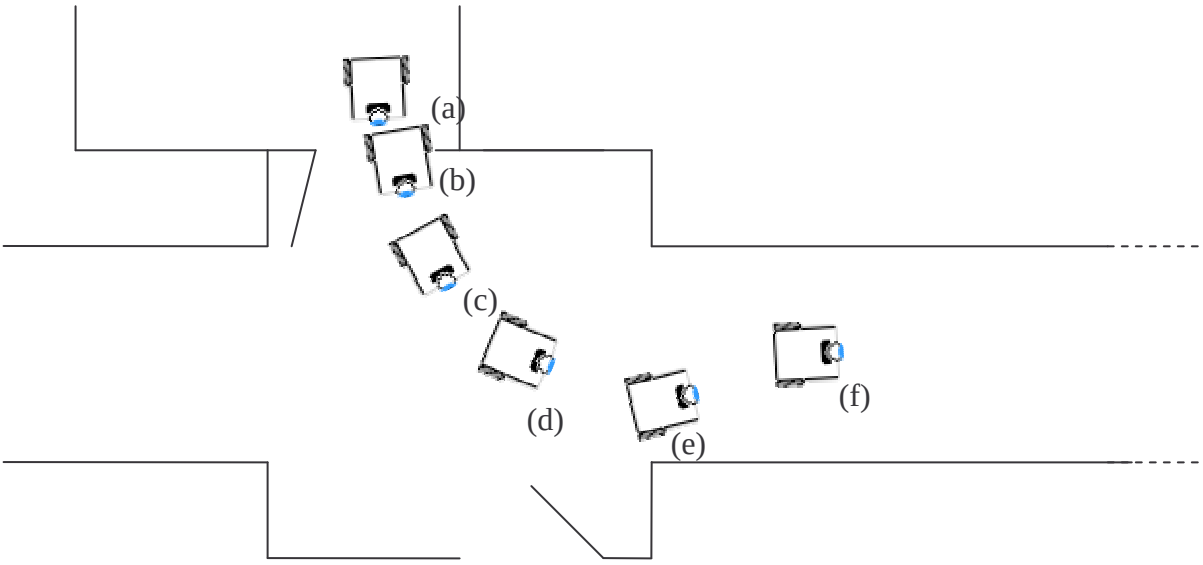
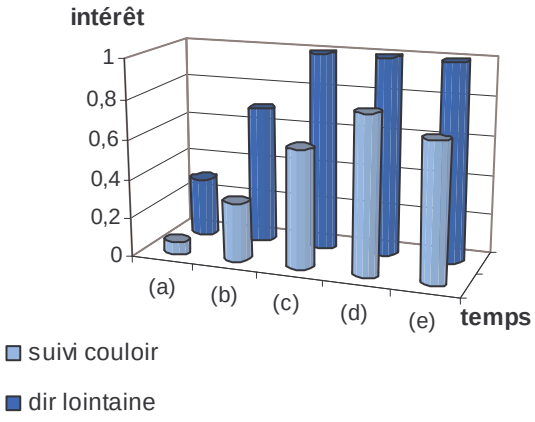


Figure 6-1-b : Trajectoire correspondante aux six photos ci-dessus.



T = (a)	Direction lointaine
T = (b)	Direction lointaine
T = (c)	Suivi de couloir
T = (d)	Suivi de couloir
T = (e)	Suivi de couloir

Figure 6-1-c : Valeurs d'intérêt et comportements associés.

Figure 7-2-a : évitement d'un obstacle simple quasi statique.
Le robot n'est pas perturbé par la présence de personnes marchant dans les couloirs.

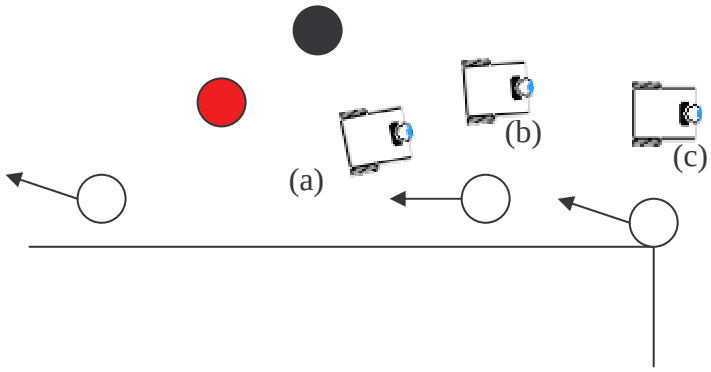


Figure 6-2-b : le robot réalise un petit écart pour s'éloigner de la personne en blanc qui marche à sa rencontre.

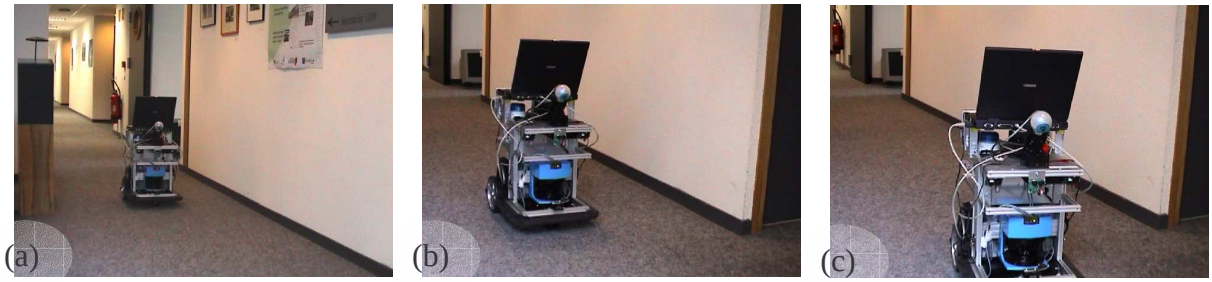
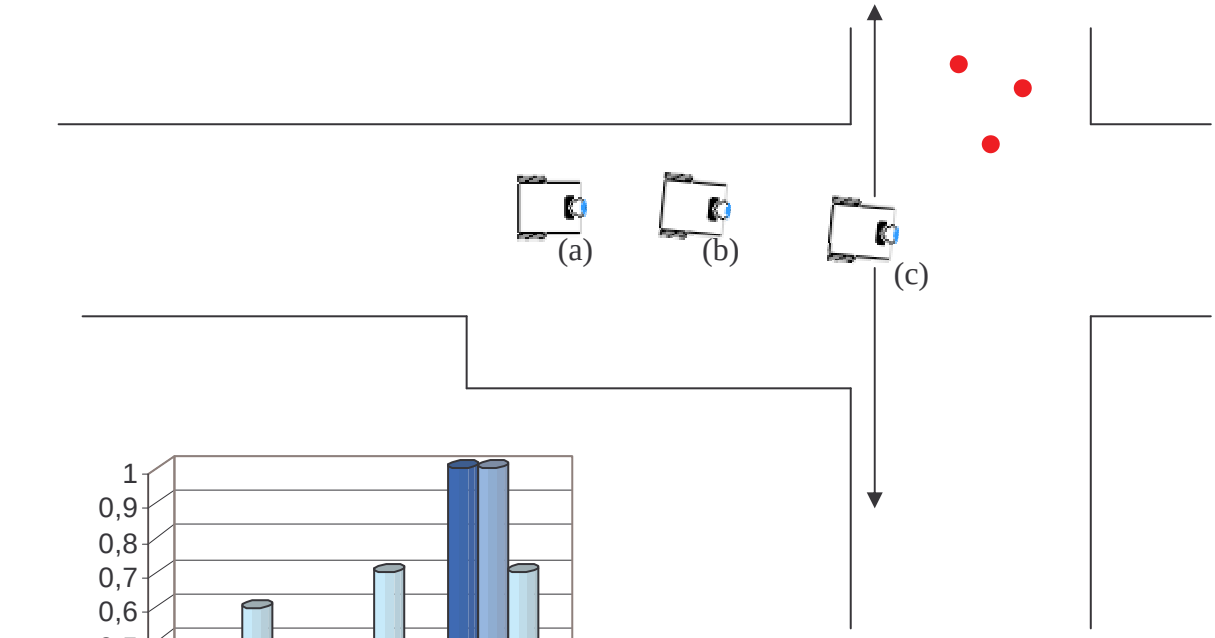


Figure 7-3-a : Arrivée dans une intersection. On étudie l'augmentation de l'intérêt du comportement « détection d'intersection ».
(les marques rouges sont représentés pour aider à la compréhension de l'expérience suivante).



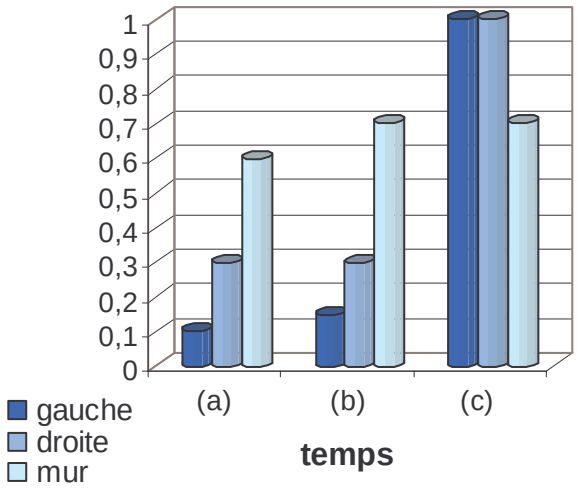
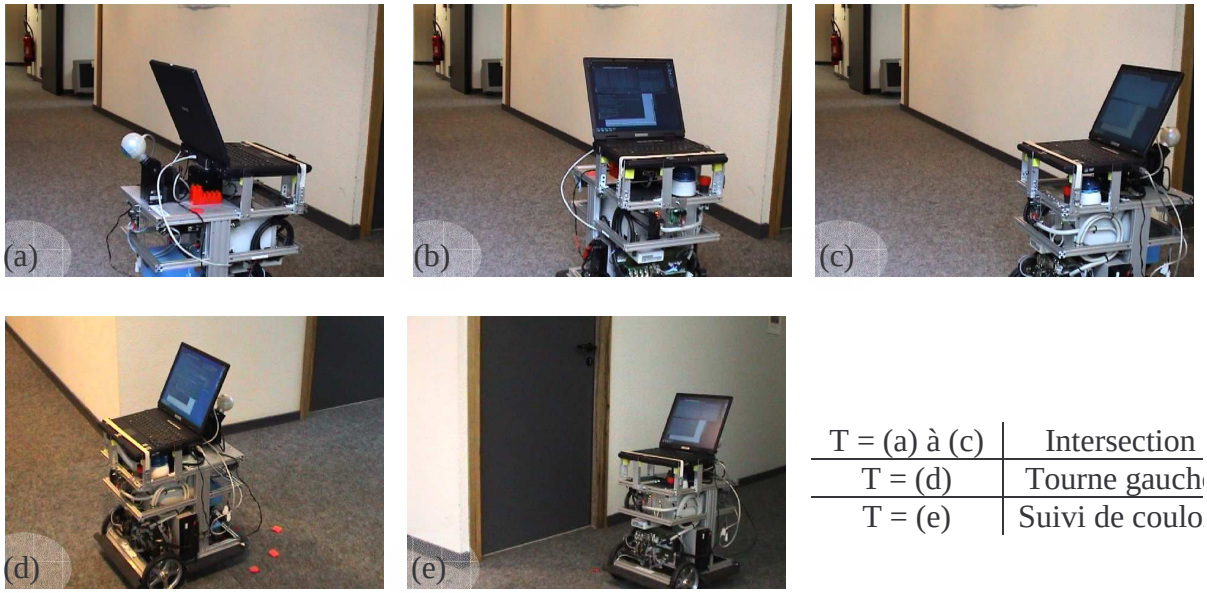


Figure 6-3-b : Trajectoire correspondante et valeurs des trois variables associées : ouverture à gauche, ouverture à droite, mur perpendiculaire.
La droite perpendiculaire est détectée bien avant de voir les ouvertures sur les cotés.



T = (a) à (c)	Intersection
T = (d)	Tourne gauch
T = (e)	Suivi de coulo

Figure 7-4 : Guidage dans l'intersection.
Le robot recherche les marques au sol (de (a) à (c)). Il choisi alors la direction gauche. Le comportement « tourne gauche » est appliqué. Le robot repasse ensuite en suivi de couloir.
Le tableau présente le comportement appliqué à chaque instant photographié.
(on se trouve dans la même intersection que dans la figure Figure 7-3).

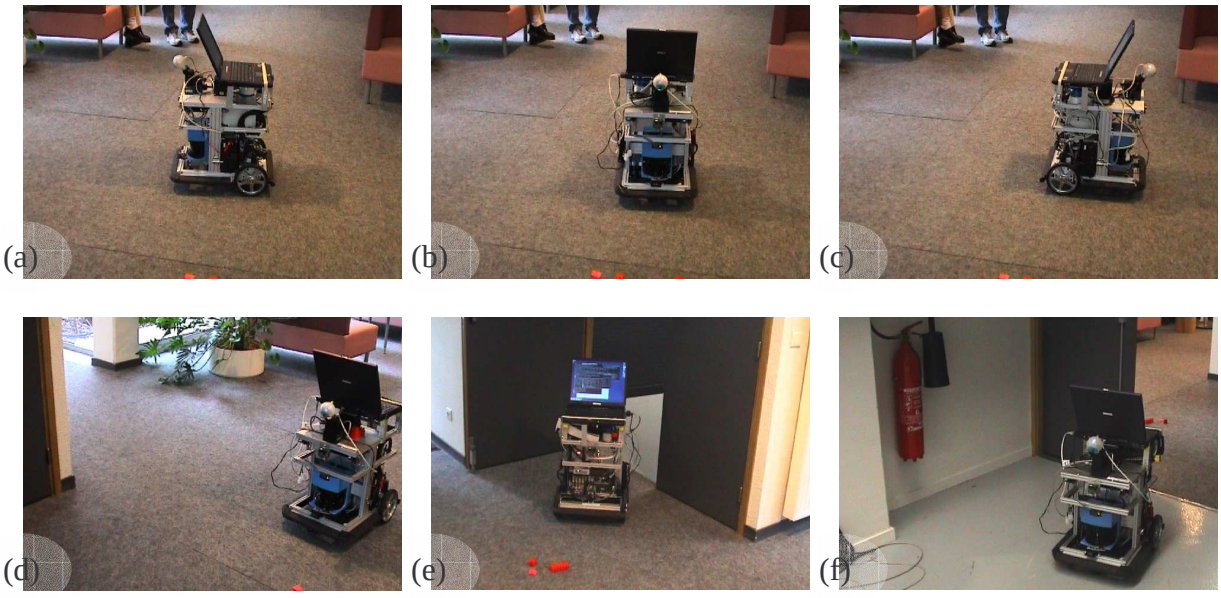


Figure 7-5 : Retour à la base.
La dernière intersection est marquée par 5 carrés rouges. Le robot passe en comportement « base » (d) et rentre dans la halle robotique (e). Il arrive en vue de sa borne de recharge (f). On considère que la tache du homing est achevée.

7.2. Partage de cartes et plus court chemin.

Nous avons présenté une exécution du retour vers la base en se servant des marques laissées lors de l'aller. Sur les photos montrant la séquence d'actions du robot, on peut voir qu'à certaines intersections, nous avons laissé plus d'une marque au sol. Nous l'avons dit plus haut, l'algorithme qui compte les marques rouges sur l'image n'est pas très efficace. Présenter plusieurs boulons au robot diminue largement la probabilité qu'il se trompe de direction.
Il semble alors logique de se demander si le guidage utilisé lors d'un premier retour est compatible avec celui disposé lors d'un second trajet pour le retour depuis **un autre point** du bâtiment. Cela revient à se demander si l'on peut **fusionner les cartes utilisées** pour chaque retour. Nous allons voir que c'est bien le cas, et que de plus, la fusion d'un grand nombre de carte semble converger vers l'arbre des plus courts chemins à but unique, où le but est bien sur la base.
Notre but n'est pas de valider en soi l'algorithme choisi pour rentrer à la base (le suivi de marque rouge), mais plutôt de montrer que notre architecture en hiérarchie, en fusionnant des comportements simples, débouche sur une implémentation puissante.

7.2.1. Partage de carte.

La fusion de plusieurs cartes est possible. En effet, marquer un chemin à l'aide des marques rouge revient à choisir un chemin sur le graphe topologique du bâtiment. Si deux chemins non disjoint (au moins un nœud en commun) vers la base sont représentés sur le même graphe, alors :

- Soit le sous-chemin à partir du premier nœud commun jusqu'à la base est identique pour les deux chemins. Dans ce cas, cela conduit à renforcer sur le graphe la partie commune ;
- Soit ce sous-chemin est différent pour les deux chemins. Comme le robot choisi une direction dans une intersection en considérant le maximum des marques rouges, en la tirant au hasard si deux couloirs en ont le même nombre.

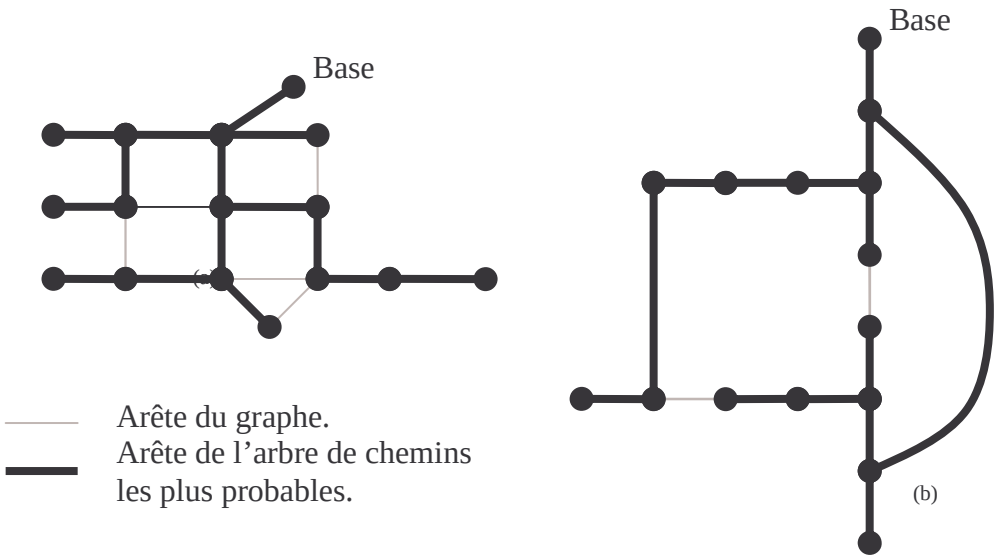


Figure 7-6 : Exemple d'arbres de chemin les plus probables obtenus après 100 aller.
(a) Le graphe de du bâtiment de l'INRIA dans lequel se déplace le robot.
(b) Un graphe quelconque plus complexe (planaire, pour faciliter sa représentation).
Les arêtes ont toutes le même poids.

De cette manière, si on considère un ensemble de trajets possibles pour le retour depuis un

point A, le chemin le plus souvent utilisé sera préféré, car il correspond aux maxima dans les intersections traversé. Si on fusionne un grand nombre de chemin vers la base, l'ensemble des chemins de plus fortes probabilités forment un arbre des chemins à but unique.

Nous avons rapidement étudié ce phénomène. Cela nécessite de réaliser de nombreux trajets d'aller et de retour. Vu la taille du bâtiment, il nous a semblé préférable de réaliser les tests sur un simulateur simple de graphes topologiques.

7.2.2. Plus courts chemins.

Le simulateur permet de réaliser des tests sur un grand nombre d'exécutions rapidement. Le robot réalise un trajet partant de la base. A chaque intersection, il choisi au hasard une direction **sans retourner sur ses pas**. Si l'intersection est une impasse, il s'arrête et rentre à la base. Sinon, le robot décide par un tirage sous une probabilité p = 1/N donnée fixe au début de l'algorithme de retourner à la base. Cela nous donne une espérance de N sur la longueur des chemins parcourus.
Nous étudions l'arbre des chemins les plus probables obtenus. Deux exemples de d'arbres sont donnés sur la Figure 7-6. On constate que ces arbres correspondent **aux arbres de plus courts chemins vers un but unique.**

7.3. Conclusion.

Le principal but de cette partie était d'étudier dans le cas réel l'évolution d'une hiérarchie de comportements. Nous avons vu dans le premier paragraphe que l'implémentation bayésienne de **cette architecture respectait les définitions données** dans le chapitre 3. L'ensemble des comportements s'articule au cours de l'expérience de manière logique. Les actions successives du robot lui permettent finalement de rejoindre la base.
La seconde partie s'est intéressée à l'algorithme de guidage du point de vue général. Nous avons vu que le choix des comportements et de la logique du guidage amenait le robot à pouvoir **fusionner sa carte courante avec celles utilisées au temps précédent**. De la même manière, cela permettrait à deux robots de partager une seule carte.
Finalement, cette **partie valide donc expérimentalement ce travail**, et montre l'intérêt de programmer un robot de manière hiérarchique en utilisant l'inférence bayésienne.

Chapitre 8.

Conclusion.

> To operate automously, mobile robots must know where there are.
> D. Fox, S. Thrun.

8.1. Contributions.

Nous avons présenté une méthode formelle pour structurer un ensemble de comportements. Notre hiérarchie de comportements garantit l'utilisation du comportement minimal capable de traiter les données de l'environnement. Cette formalisation a alors été rattachée à la théorie des séquences temporelles bayésiennes, et une méthode générale d'implémentation a été proposée.
Nous avons ensuite présenté un exemple d'application de ce formalisme. Notre robot répond au problème du *homing* en utilisant uniquement des méthodes de navigation locale. L'utilisation de la hiérarchie de comportements permet d'une part que le robot s'approche de la base en suivant le couloir dans lequel il se trouve, en changeant de comportement quand celui-ci ne suffit plus, c'est-à-dire dans une intersection. Elle assure aussi que le robot passera en recherche visuelle quand il sera suffisamment proche de la base. Enfin, en imposant un comportement minimal, elle garantit que le robot ne se trouvera pas dans une situation dont il ne sait pas se sortir.
La contribution de ce travail est donc double :

- D'une part, **théorique**, avec la formalisation d'une hiérarchie de comportements qui ouvre une nouvelle voie de recherche pour résoudre les problèmes si délicats du développement incrémental et du passage à l'échelle en robotique ;
- D'autre part, **pratique**, en ayant conduit jusqu'au bout l'application de cette méthodologie à un cas réel pour en expérimenter la validité.

8.2. Perspectives.

8.2.1. Développements pratiques.

Nous avons proposé une implémentation n'utilisant que des techniques de navigation locale. En ce sens, le robot arrive à rejoindre sa base *sans jamais savoir où il se trouve*. Malgré la simplicité des comportements de base choisis, la stratégie globale permet de rejoindre le but avec de bon taux de réussite. Il serait très intéressant de mettre en œuvre ce formalisme en œuvre sur des stratégies de plus haut niveau. En particulier, il nous paraît intéressant d'essayer de fusionner des cartes métriques et topologiques en utilisant notre méthode.

8.2.2. Développements théoriques.

Notre robot navigue donc sans représentation de l'espace. Les comportements qu'il applique et la manière de les ordonner constituent la seule carte qu'il utilise. Il nous semble intéressant de relier la méthode formelle que nous venons de proposer à la théorie générale des cartes bayésiennes présentée dans [{DIARD03}. Cette théorie propose un formalisme général des modélisations probabilistes de l'interaction entre un robot et son environnement. L'auteur y

propose une formalisation de la notion de carte ainsi que deux opérateurs permettant d'assembler des cartes pour en créer de nouvelles.
Pour notre part, nous proposons une structure pour assembler des comportements, en particulier pour la navigation. Un rapprochement entre notre méthode et cette théorie globale paraît donc prometteur.

Bibliographie.

{ARGYROS02} A.A. Argyros, C. Bekris, S. Orphanoudakis, « **Robot homing based on corner tracking in a sequence of panoramic images** », Computer Vision and Pattern Recognition Conference (CVPR '2001), 11-13 December 2001, Hawaii, USA.

{BAILEY01} T. Bailey, E. Nebot, « **Localisation in large-scale environnements** », robotics and autonomous systems, vol. 37, p. 261-281, 2001.

{BESSIERE98} P. Bessière, E. Dedieu, O. Lebeltel, E. Mazer, K. Mekhnacha, « **Interprétation ou description (II) : fondement mathématiques de l'approche F+D** », Intellectica, revue pour l'association de la recherche cognitive, 1998/1-2, 26-27, p. 313-336, 1998.

{BROOKS91} R.A. Brooks, « **Intelligence Without Reason** », Proceedings of 12th International Joint Conference on Artificial Intelligence, Sydney, Australia, August 1991, p. 569-595.

{CHATILA85} R. Chatila, J.P. Laumond, « **Position referencing and consistent world modeling for mobile robot** », Proc. IEEE International Conference on robotics and automation, p. 138-145, 1985.

{COLLETT96} T.S. Collett, « **Insect navigation en route to the goal: multiple strategies for the use of landmarks** », The journal of experimental biology, 199, 227-235 (1996).

{COOMBS93} D. Coombs, K. Roberts, « **Centering behavior using peripheral vision** », Proceedings of IEEE conference on computer vision and pattern recognition, Los Alamitos, CA, 1994, p. 2272-2277.

{COUE00} C. Coue, « **Programmation bayésienne des robots: poursuite d'une cible fuyante par un robot mobile** », thèse de DEA, Institut Nationnal Polytechnique de Grenoble, Grenoble, France, Juin 2000.

{COUE01} C. Coué, P. Bessière, « **Chasing an elusive target with mobile robot** », Proc. IEEE International conference on intelligent robot and system, Hawaii, USA, October 2001, p. 256-260.

{COUE03} C. Coué, C. Pradalier, C. Laugier, « **Bayésian programming for multi-target tracking : an automotive application** », à paraître.

CROWLEY89} J.L. Crowley, « **World modeling and position estimation for a mobile robot using ultrasonic ranging** », AAAI workshop on spatial reasoning and multisensor fusion, Kaufmann Press, 1987.

{DIARD03} J. Diard, « **La carte bayésienne** », Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France, Janvier 2003.

{ELFES89} A. Elfes, « **Using occupancy grids for mobile robot perception and navigation** », Computer 22, No 6, June 1989: 46-57.

{FRANZ98} M.O. Franz, B Schölkopf, H.A. Mallot, H.H. Bülthoff, « **Learning view graph for robot navigation** », Autonomous robots 5, 111-125 (1998).

{FRANZ00} M.O. Franz, H.A. Mallot, « **Biometric robot navigation** », Robotics and Autonomous System 30 (2000) 133-153.

{JONES97} S.D. Jones, C. Andresen, J.L. Crowley, « **Appearence based process for visual navigation** », Proc. IEEE international conference on robots and systems, p. 551-557, Sept. 1997.

{KUIPERS89} B. Kuipers, Y.T. Byun, « **A robust, qualitative method for robot spatial learning** », Proc. AAAI-88, p. 774-779.

{KUIPERS91} B. Kuipers, Y.T. Byun, « **A robot exploration and mapping strategy based on semantic hierarchy of spatial representations** », Robotics and autonomous systems, vol. 8, No 1, p. 47-69, 1991.

{KUIPERS00} B.J. Kuipers, « **The Spatial Semantic Hierarchy** », Artificial Intelligence 119: 191-233.

{LAMBRINOS00} D. Lambrinos, R. Möller, T. Labhart, R. Pfeifer, R. Wehner, « **A Mobile Robot Employing Insect Strategies for Navigation** », Robotics and Autonomous Systems. 30, 39-64, 2000.

{LEBELTEL99} O. Lebeltel, « **Programmation bayésienne des robots** », Thèse de doctorat, Institut Nationnal Polytechnique de Grenoble, Grenoble, France, Octobre 99.

{LEONARD91} J.J. Leonard, H.F. Durrant-Whyte, « **Mobile robot localization by tracking by tracking geometric beacons** », IEEE Transactions on robotics and automation, vol. 7, No. 3, June 1991.

{LEVITT90} T.S. Levitt, D.T. Lawton, « **Qualitative navigation for mobile robots** », Artificial Inteligence 44 (1990) 305-360.

{LORIEUX03} J. Lorieux, « **Détection de voitures** », Thèse de DEA, Institut National Polytechnique de Grenoble, Grenoble, France, Juin 2003.

{MACFARLAND2001} D. McFarland, « **Le comportement animal, psychobiologie, éthologie et évolution** », DeBoeck Université, 1998.

{MARTIN02} M.C. Martin, « **Genetic programming for real world robot vision** », Proceedings of the IEEE intl. conference on intelligent robots and systems, EPFL, Switzerland, octobre 2002

{MENG93} M. Meng, A.C. Kak, « **Mobile robot navigation using neural networks and nonmetrical environment models** », IEEE control systems, p. 30-39, octobre 1993.

{MOLLER00} R. Möller, D. Lambrinos, R. Pfeifer, T. Labhart, R. Wehner, « **Modeling ant navigation with an autonomous agent** », From Animals to Animats 5, Proceedings of SAB'98, MIT Press, Cambridge, MA, 1998, p. 185-194.

{NILSSON69} N.J. Nilson, « **A mobile automation: an application of artificial intelligence techniques** », Proc. of the first IJCAI, pp 509-520, 1969.

{PAPI92} F. Papi, « **Homing and related phnomena** », de « Animal homing », Chapman & Hall, London, 1992, p. 687-695.

{SCHARSTEIN00} D. Scharstein, A. Briggs, « **Fast recognition of selfsimilar landmarks** », Workshop on Perception for Mobile Agents, pages 74--81, June 1999.

{SHARP98} T. Sharpe, B. Webb, « **Simulated and situated models of chemical trail following in ants** », From Animals to Animats 5, Proceedings of SAB'98, MIT Press, Cambridge, MA, 1998. p. 195-204.

{TAPUS02} A. Tapus, « **Markov Decision Processes Use for Mobile Robot Actions Planning and Execution** », Thèse de DEA, Institut National Polytechnique de Grenoble, Grenoble, France, Juin 2002.

{THRUN96} S. Thrun, A. Bücken, « **Learning maps for indoor mobile robot navigation** », Technical Report CMU-CS-96-121, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1996.

{THRUN98} S. Thrun, « **Learning metric-topological maps for indoorsmobile robot navigation** », Artificial Inteligence, 99(1), 41-76, 1998.

{THRUN00} S. Thrun, D. Fox, W. Burgard, F. Dellaert, « **Robust monte carlo localization for mobile robots** », Activity repport No CMU-CS-00-125, School of computer science, Carnegie Mellon University, Pittsburg, PA 15213, April 2000.

{THRUN02} S. Thrun, « **Robot mapping : a survey** », technical repport No CMU-CS-02-111, Carnegie Mellon University, February 2002.

{TOMATIS01a} N. Tomatis, I. Nourbakhsh, K.O. Arras, R. Siegwart, « **An Hybrid Approach for Robust and Precise Mobile Robot Navigation with Compact Environment Modeling** », Proc. of the IEEE International Conference on Robotics and Automation, Seoul, Korea, May 2001.

{TOMATIS01b} N. Tomatis, I. Nourbakhsh, K.O. Arras, R. Siegwart, « **Simultaneous Localization and Map Building: A Global Topological Model with Local Metric Maps** », Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, USA, October/November 2001.

{TRULLIER96} O. Trullier, S.I. Wiener, A. Berthoz, J.-A. Meyer, « **Biologically based artificial navigation systems : Review and prospects** », Progress in Neurobiology 51 (1997), p. 483-544.

{TSAKIRIS00} D. Tsakiris, A.A. Argyros, « **Corridor Following by Nonholonomic Mobile Robots Equipped with Panoramic Cameras** », IEEE Mediterranean Conference on Control and Automation , Invited Session, « Sensor-based Control for Robotic Systems », Patras, July 2000.

{VICTORINO02} A.C. Victorino, **« La commande référencée capteur: une approche robuste au problème de navigation, localisation et cartographie simultanées pour un robot d'interieur** », Thèse de doctorat, Université de Sophia Antipolis, Nice, France.

{WEBER99} K. Weber, S. Venkatesh, M. Srinivasan, **« Insect-inspired robotic homing** », Adaptative behavior, 1999, 7 (1), p. 65-97.

{WEBER97} K. Weber, S. Venkatesh, M.V. Srinivasan, « **Insect inspired behaviours for the autonomous control of mobile robots** », From Living Eyes to Seeing Machines, Oxford University Press, Oxford, 1997, p. 226-248.

{YGUEL03} M. Yguel, « **Maximum d'entropie : application à la robotique** », Thèse de DEA, Institut National Polytechnique de Grenoble, Grenoble, France, Juin 2003.

Annexes

Annexe A. Exemples de modèles bio-inspirés appliqués à la robotique.

A.1. Méthodes de navigation locale.

A.1.1. Recherche au hasard.

Le premier comportement de la hiérarchie sont trop simples pour avoir fait l'objet de travaux d'imitation par les laboratoires de robotiques. D'après {FRANZ00} pourtant, cette stratégie mériterait d'être plus étudiée, au moins comme dernière solution lorsque les autres stratégies ont échoué. Cette réflexion est d'autant plus vraie que des études poussées ont été menées pour étudier cette stratégie du hasard, en particulier chez les isopodes du désert. Les autres comportements sont par contre plus utilisés.

A.1.2. Suivi d'une direction et intégration du chemin parcouru.

Le premier niveau de stratégie (suivi d'une direction) pose deux problèmes : trouver et suivre une direction, d'une part, et mesurer la distance parcourue de l'autre. Le Sahabot II utilise la polarisation de la lumière du soleil comme boussole pour se diriger en ligne droite ({LAMBRINOS00}). L'extraction de la direction est directement donnée à partir de la mesure capteur. Les auteurs de {SHARP98} utilisent des capteurs chimiques pour suivre une trace de vapeur d'alcool. L'intérêt de leur démonstration est double puisqu'en plus d'intéresser la robotique, il valide une théorie éthologique[1] (Figure A-1). Pour mesurer la distance parcourue, les robots utilisent classiquement l'odométrie, c'est à dire le nombre de tour de roue. Le problème réside, on l'a vu, dans l'accumulation des erreurs. Une solution a été proposée pour le robot Sahabot dans

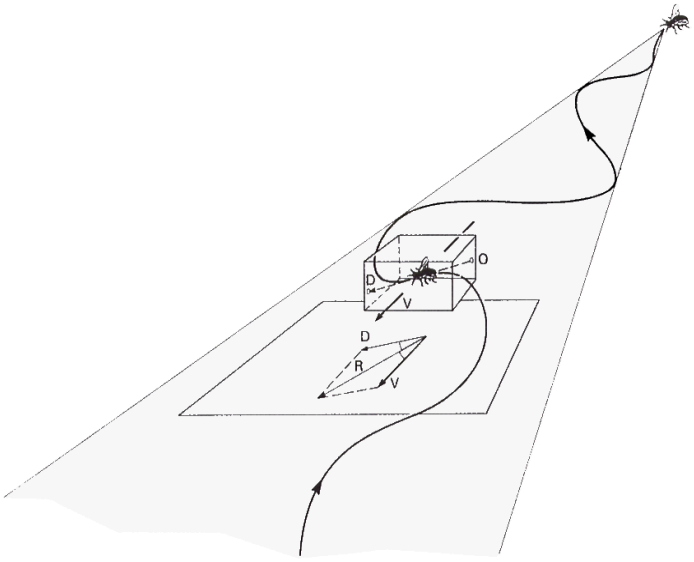


Figure A-1 : A la manière du papillon de nuit volant contre le vent en réponse à une phéromone libérée par une femelle, le robot présenté dans {SHARP98} oscille à la recherche de la route tracée par les vapeurs d'alcool.
Source: {MACFARLAND2001}, p. 271.

[1] Ethologie : Science des mœurs, plus particulièrement, étude des comportements spontanés des animaux dans leur habitat naturel ou dans un environnement proche de celui-ci.

{LAMBRINOS00} : le robot corrige ses erreurs d'orientation en utilisant le compas solaire dont on a parlé plus haut. Il utilise au final une fusion entre son odométrie et sa boussole pour estimer sa position.

A.1.3. Suivi d'amer.

Dans {COUE00}, l'auteur implémente un comportement de suivi d'une cible de couleur par un robot possédant une caméra mobile. Il s'agit (même si ce n'est pas à proprement parler du but de ce travail) de la reproduction de la stratégie de suivi d'un amer.

A.1.4. Guidage par points de référence.

Dans {WEBER99} et dans {MOLLER00}, le comportement de la fourmi du désert 'Cataglyphis bicolor' est imité à l'identique, d'après les travaux de T.S. Collett (entre autre). Ces fourmis ne peuvent pas utiliser de phéromones à cause de la chaleur de leur habitat naturel, le Sahara. Elles utilisent une stratégie particulière: elles cherchent à mettre en correspondance ce qu'elles voient avec une image mentale de l'horizon autour de la fourmilière. Pour cela, elles simplifient l'image panoramique de ce qui les entourent en un anneau alternant zones sombres et zones claires, comme montré Figure A-2. A tout instant, la fourmi met en correspondance les deux images, celle du but, et ce qu'elle voit, et calcul la direction à suivre, comme expliqué sur la Figure A-3. Ce comportement correspond exactement au guidage par points de référence.

A.2. Recherche d'un chemin.

A.2.1. Association reconnaissance – action.

Les comportements de recherche de chemin sont plus courants et plus étudiés en robotique classique. Le premier comportement de ce groupe, qui consiste à suivre la route tracée par un graphe topologique linéaire, est pourtant essentiellement trouvé dans des études biomimétiques, par exemple celle menée dans {ARGYROS02}. Le robot est dirigé à l'aide d'images prises par une caméra panoramique lors d'un premier trajet (les auteurs se placent dans le cadre d'un retour à la base). Il détecte des zones caractéristiques de l'image (good features to track) et enregistre leur cycle de vie, c'est à dire leur date de première détection et leur date de disparition. Au retour, le robot calcule la vitesse qu'il doit adopter en fonction des cibles qu'il détecte et des cycles de vie qu'il a enregistré.

A.2.2. Graphe topologique.

La navigation topologique est largement utilisée en environnement structuré, typiquement l'intérieur d'un bâtiment où deux lieux sont séparés par un couloir, par exemple dans {KUIPERS91}. Dans {FRANZ98}, les auteurs exposent une méthode pour tracer automatiquement un graphe en environnement extérieur. Le robot (un Khepera) apprend à se localiser dans une ville miniature (Figure A-4). Les différents lieux sont reconnus en comparant à une base de donnée d'image la vue courante prise à l'aide d'une caméra panoramique (un capteur apprécié dans la robotique bio inspirée). La stratégie de navigation locale entre deux noeuds du graphe utilise un guidage par points référence, expliqué plus haut à propos des articles {WEBER99} et {MOLLER00}.

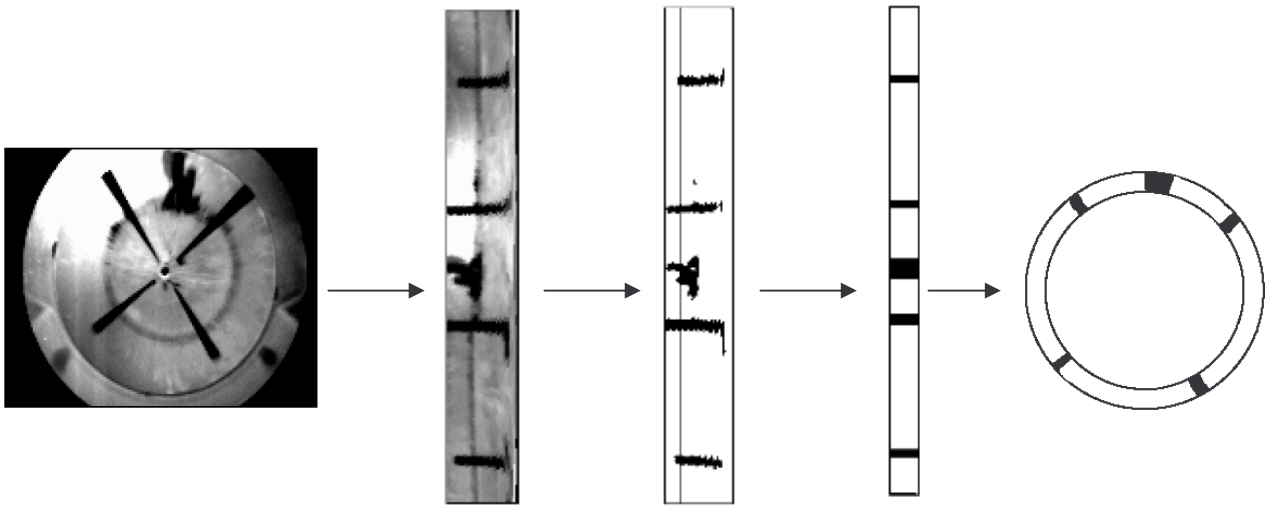


Figure A-2 : Pré-traitement de l'image effectué par le robot Sahabot II, imitant la vision de la fourmi du désert.
(a) image de la cméra panoramique.
(b) Projection depuis le repère en coordonnées polaires.
(c) Seuillage.
(d) Extraction d'un vecteur linéaire simple.
(e) Représentation du vecteur sous forme circulaire.
Source: {MOLLER00}.

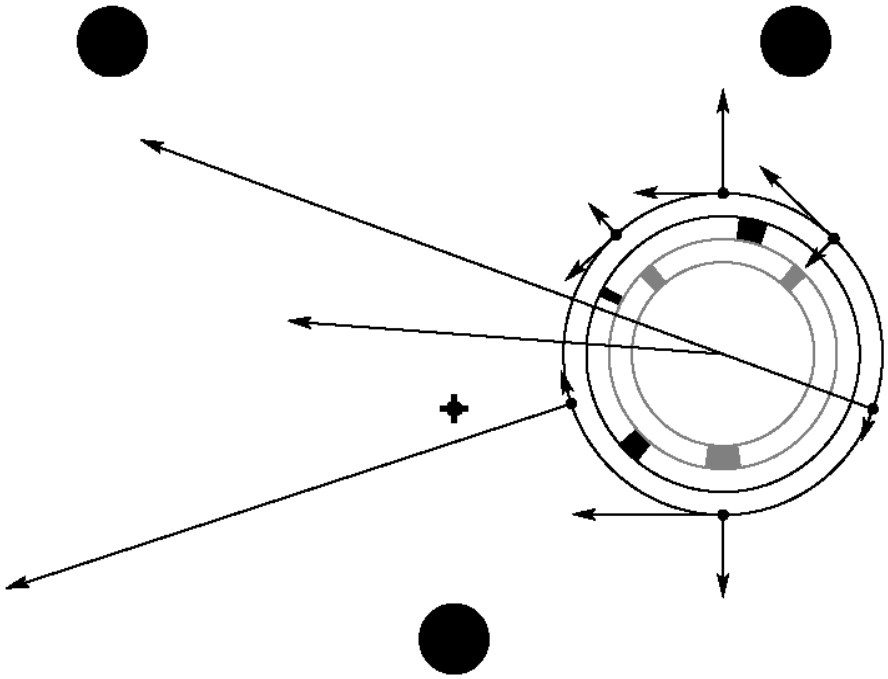


Figure A-3 : L'image de référence (anneau du centre, en gris) est comparée à l'image courante (anneau externe, en noir). Chaque couple de secteurs mis en correspondance donne lieu à un vecteur de translation et un vecteur de rotation. La direction de la fourmilière est obtenue en faisant la moyenne des trois.
Source: {MOLLER00}.

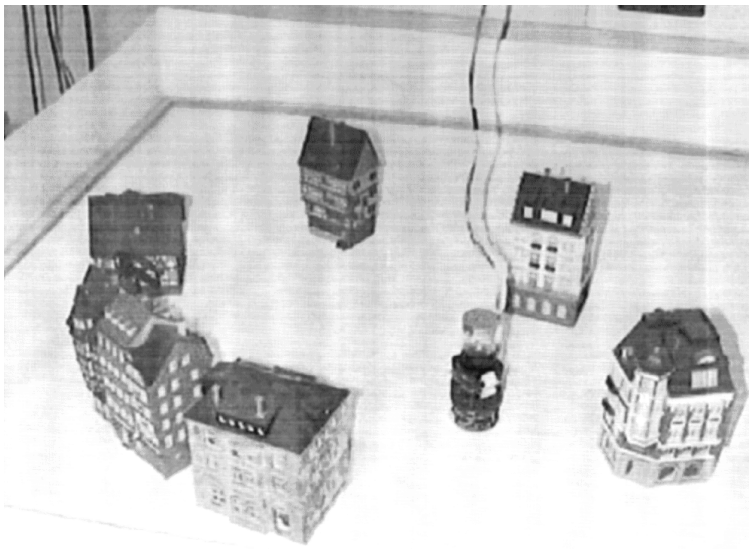


Figure A-4 : Navigation topologique dans un environnement non structuré : une ville miniature à l'echelle du Khepera.
Source: {FRANZ98}.

A.2.3. Navigation omnisciente.

Cette dernière méthode consiste à construire à mémoriser une structure stockant les distances entre tous les points de la carte. C'est typiquement ce qui est fait dans les cartes utilisées en navigation classique. Le robot peut ainsi trouver des raccourcis et déduire de nouveaux chemins en territoire inconnu.

A.3. Où placer les méthodes de robotiques classiques dans notre hiérarchie ?

Les méthodes que nous avons présentées dans la partie 2.1 peuvent aussi se ranger dans la hiérarchie bio-inspirée. Quelques exemples sont donnés rapidement ici :

- les grilles d'occupations présentées dans {ELFES89}, et reprise dans {THRUN97} correspondent à ce que nous avons appelé navigation omnisciente.
- Dans {KUIPERS91}, les auteurs présentent une formalisation des graphes topologiques, qui correspondent évidement à la navigation topologique.
- La navigation par association reconnaissance - action permet de tracer une route simple allant d'un point à un autre. C'est la méthode qui est utilisée dans {JONES97}: à une image enregistrée par la caméra est associée une action, à l'aide du mesure de similarité. Nous avions présenté la méthode avec plus de détail au début du paragraphe 2.1.2.
- Dans {BAILEY01}, les auteurs proposent une méthode utilisant une carte globale sous forme d'un graphe topologique, dont les différents noeuds pouvaient être différenciés à l'aide de primitives géométriques, comme dans {CROWLEY89}. On peut ramener cette méthode à une navigation globale topologique, utilisant un guidage par points de référence comme méthode de navigation locale.

Au final, de nombreuses méthodes peuvent se décrire comme une architecture de contrôle bio inspirée. On peut aussi trouver des méthodes classiques reprenant les niveaux de la navigation locale, mais on quitte alors le domaine de la localisation pour rentrer dans celui de la robotique réactive. Nous n'avons pas présenté de telles méthodes car cela sort de notre sujet.

Annexe B. Trois comportements minimaux de suivi de couloir.

Nous présentons ici trois comportements minimaux pour suivre un couloir (minimaux au sens du nombre de données capteur utilisées). Ces trois comportement justifie l'utilisation d'un comportement plus complexe dans notre hiérarchie de contrôle. En effet, les comportements obtenus ne sont pas suffisant pour respecter les spécifications imposées par la hiérarchie définie dans le chapitre 3.

B.1.1. A l'aide de l'odométrie.

a) Présentation.

Pour cette première approche, on concidère que le robot se situe au départ au milieu du couloir (hypothèse 1). Il n'a donc pas besoin de savoir s'y placer, mais seulement d'y rester. Si on défini un repère dont l'origine se situe à l'emplacement initial du robot (cf. Figure B-1), ce comportement s'écrit :

$$\begin{cases} Y = 0 & (1) \\ \theta = 0 & (2) \end{cases}$$

Le but est ensuite de faire croître X, qui correspond à l'avancée dans le couloir.

b) Spécifications.

- **Variables:**

  Vrot: vitesse de rotation du robot.
  Y : distance au milieu du couloir donné par l'odométrie.
  θ: position angulaire donnée par l'odométrie.

- **Décomposition:**

$$P(Vrot, Y, \theta) = P(Y) \times P(\theta) \times P(Vrot \mid \theta, Y)$$

- **Forme paramétrique:**

P(Y), P(θ): nous n'avons pas d'a priori sur ces variables. Nous associons donc à ces termes une loi uniforme.

$$\mathbf{P(Vrot \mid \theta, Y)} = \mathcal{G}(\omega(Y) + \omega(\theta), \sigma(Y) + \sigma(\theta)).$$

On adopte ici une moyenne entre deux comportements : celui qui conduit le robot à se positionner sur le milieu du couloir (respectant la commande (1)), et celui qui recentre le robot dans l'axe du couloir (commande (2)). Le comportement adopté correspondra à la commande qui requiert la vitesse la plus élevée. Les deux comportements se renforcent si ils vont dans le même sens. L'incertitude (variance de la loi normale) augmente si l'incertitude sur l'un des

comportements augmente.

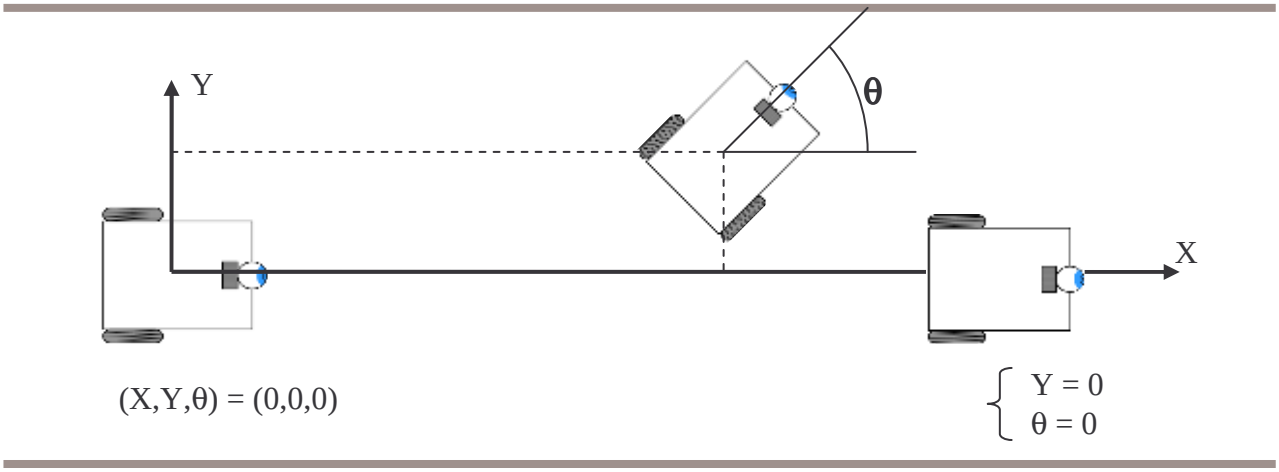


Figure B-1 : On définit un repère dont l'origine correspond à l'emplacement initial du robot. Le robot navigue sur la droite (Y=0, θ=0).

c) Identification.

Le seul a priori à spécifier est celui sur la vitesse de rotation. Plus le robot s'éloigne du centre du couloir sur la droite (Y > 0), plus il doit tourner vite à gauche (vitesse de rotation positive). De même pour θ > 0. On choisit :

$$\omega(Y) = K_Y.Y^3, \qquad \text{avec } K_Y > 0.$$

$$\omega(\theta) = K_\theta.\theta^3, \qquad \text{avec } K_\theta < 0.$$

Pour ce comportement, on va choisir la valeur de Vrot qui maximise la distribution de probabilité. La valeur de σ(Y) et σ(θ) importe donc peu. On choisit σ(Y) = σ(θ) = 1.

d) Question.

La question à poser est bien évidemment :

$$P(Vrot \mid Y, \theta).$$

e) Evaluation du comportement.

Le robot suit facilement la ligne (Y=0). Le comportement obtenu est stable sur cette ligne. Cette stabilité est renforcée grâce au cube dans le calcul de ω(Y) et ω(θ). Les premiers essais, qui utilisaient des vitesses de rotation linéaires en θet Y tendaient à faire osciller légèrement le robot autour de la position d'équilibre. En cas de perturbation de la position d'équilibre, le robot retrouve facilement la ligne (Y=0) et la stabilité désirée.
Par contre, le robot ne suit ***absolument*** pas le centre du couloir. En effet, il utilise son odométrie pour se recentrer, c'est à dire l'hypothèse (1) (définie en 4.1.3.a). Or, il est impossible en pratique d'aligner au départ le robot sur l'axe du couloir avec suffisamment de précision. Au bout d'une dizaine de mètres, le robot a tellement divergé qui rentre en contact avec le mur.
Ce comportement, un peu simpliste et très limité, nous a tout de même permis d'introduire la démarche que nous allons adopter dans la suite, ainsi qu'un « plus mauvais comportement » auquel comparer tous les autres.

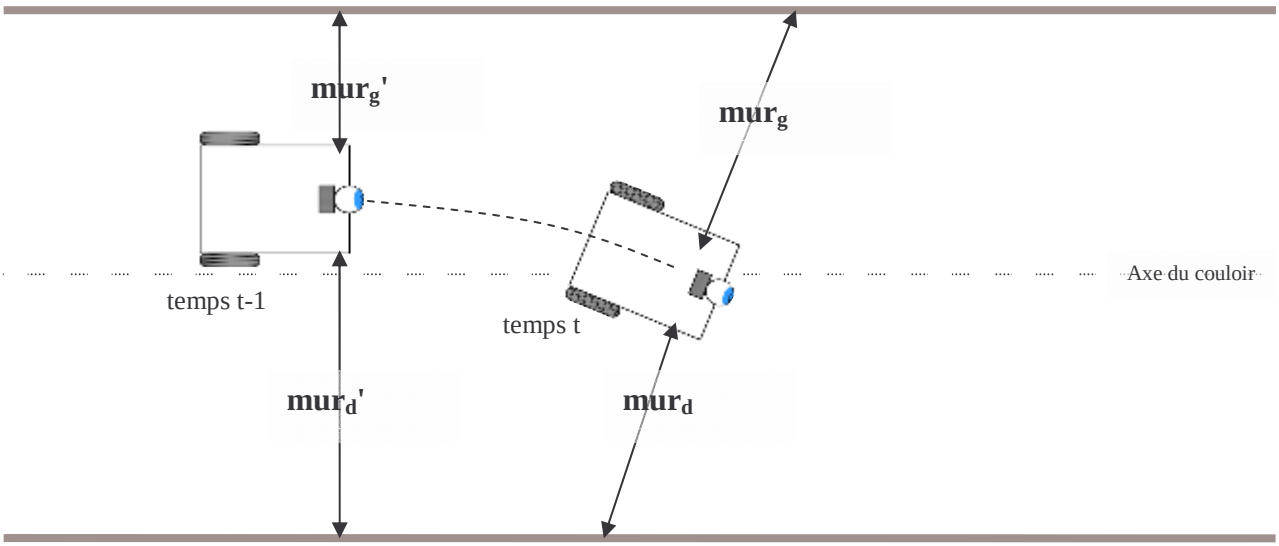


Figure B-2 : On note mur$_g$ et mur$_d$ les mesures capteurs au temps courant à +90° et 90°. On note mur$_g$' et mur$_d$' ces mesures prises au temps précédent

B.1.2. Par la distance aux murs.

a) Présentation.

Le robot se repère par rapport aux murs. On note mur$_d$ la distance donnée par le capteur laser sur la droite, et mur$_g$ la distance lue par le capteur sur la gauche (cf. Figure B-2). Il suffit que le robot avance en maintenant ces deux valeurs égales. La loi de commande s'écrit alors simplement « mur$_d$ - mur$_g$ = 0 ».
Cette loi de commande spécifie au robot de se placer au centre, mais pas dans l'axe du couloir. Sur la Figure B-2 par exemple, le robot se trouve à égale distance des deux murs (au temps t). La commande lui impose donc une vitesse de rotation nulle alors que l'on désire qu'il se repositionne dans l'axe. Dans ce cas, le robot quittera le milieu du couloir au temps suivant, et tournera sur la gauche seulement à ce moment. On peut montrer que cette loi de commande n'est pas stable et conduit le robot à osciller.
On va donc chercher à contraindre le robot à une stabilité par rapport aux variables mur$_d$ et mur$_g$. On note mur$_d$' et mur$_g$' les distances lues au temps précédent (cf. Figure B-2). On impose au robot de garder ces valeurs constantes. On obtient les commandes suivantes :

$$\begin{cases} \text{mur}_d - \text{mur}_g = 0 & (1) \\ \text{mur}_d - \text{mur}_d' = 0 & (2) \\ \text{mur}_g - \text{mur}_g' = 0 & (3) \end{cases}$$

b) Spécification

- **Variables:**

Vrot: vitesse de rotation du robot.
mur$_d$', mur$_g$': distances lues par le capteur laser au temps précédent.
mur$_d$', mur$_g$': distances lues par le capteur laser au temps courant.

- **Décomposition**:

$$P(Vrot, mur_d', mur_g', mur_d, mur_g) = \left| \begin{array}{l} P(mur_g') \times P(mur_d') \\ \times P(mur_g \mid mur_g) \times P(mur_d \mid mur_d') \\ \times P(Vrot \mid mur_g, mur_d, mu_g', mur_d) \end{array} \right.$$

- **Forme paramétrique:**

$P(mur'_d)$, $P(mur'_g)$: Pas d'a priori sur ces variables. On choisit des lois uniformes.
$P(mur_d \mid mur_d')$, $P(mur_g \mid mur_g')$: Les distances aux murs au temps courant dépendent des distances au temps précédent. Mais ils dépendent aussi de la vitesse de translation du robot, du temps écoulé depuis les dernières mesures, de l'angle du robot par rapport à l'axe du couloir ... Autant de variables inconnues. Le plus simple serait de spécifier pour ces distributions des lois normales centrée autour des mesures du temps précédent, mais se serait une approximation trop forte. On choisit donc de les représenter par un histogramme, rempli grâce à un apprentissage non supervisé réaliser en cours d'exécution.

$$P(Vrot \mid mur_g, mur_d, mur_g, mur_d) = \zeta(\omega(mur_g - mur_d) + \omega(mur_g - mur_g') + \omega(mur_d - mur_d'), 1)$$

On donnera les expressions des ω dans le paragraphe suivant (Identification). Comme pour le programme précédent, on fait une moyenne entre trois comportements, chacun correspondant à une des lois de commande. Pour simplifier, on laisse l'incertitude (variance) constante.

c) Identification.

Comme nous l'avons dit plus haut, on rempli l'histogramme à la volée.
On doit ensuite spécifier les fonctions ω. De la même manière que dans le paragraphe précédent, on choisit :

- $\omega(mur_g - mur_d) = K_1 \times (mur_g - mur_d)$, $K_1 > 0$.

Quand le robot est plus proche du mur gauche, $mur_g - mur_d < 0$. La vitesse appliquée est négative, le robot tourne à droite.

- $\omega(mur_g' - mur_g) = K_2 \times (mur_g - mur_g')$, $K_2 > 0$.

Quand le robot se rapproche du mur gauche, $mur_g - mur_g' < 0$. Le robot tourne à droite.

- $\omega(mur_d' - mur_d) = K_3 \times (mur_d' - mur_d)$: $K_3 = K_2$.

Quand le robot se rapproche du mur droit, $mur_d' - mur_d > 0$. Le robot tourne à gauche.

d) Question.

On connaît les distances aux murs, la question à poser est donc :

$$P(Vrot \mid mur_g, mur_d, mur_g', mur_d')$$

e) Evaluation du comportement.

La première constatation est que le robot est très sensible à de légères variations de la configuration du couloir. Cela est du au fait que nous ne considérons qu'une mesure du capteur laser. Ce problème se résout facilement en utilisant la moyenne des mesures capteurs sur un angle de 5°. C'est ce que nous avons fait dans la suite. Malgré cela, le robot reste instable. Dans les cas simples, le robot suit le couloir de manière correcte. Mais il est perturbé par les changements de largeur du couloir devant les bureaux. De plus, une porte ouverte sur un bureau lui fait faire un écart important. Enfin, le comportement devient très incertain lorsque le robot arrive dans une intersection, alors que c'est à ce moment qu'on désirait un comportement sur. Il nous faut donc définir un comportement utilisant mieux les données en provenance du laser.

B.1.3. En suivant la direction la plus lointaine.

a) Présentation.

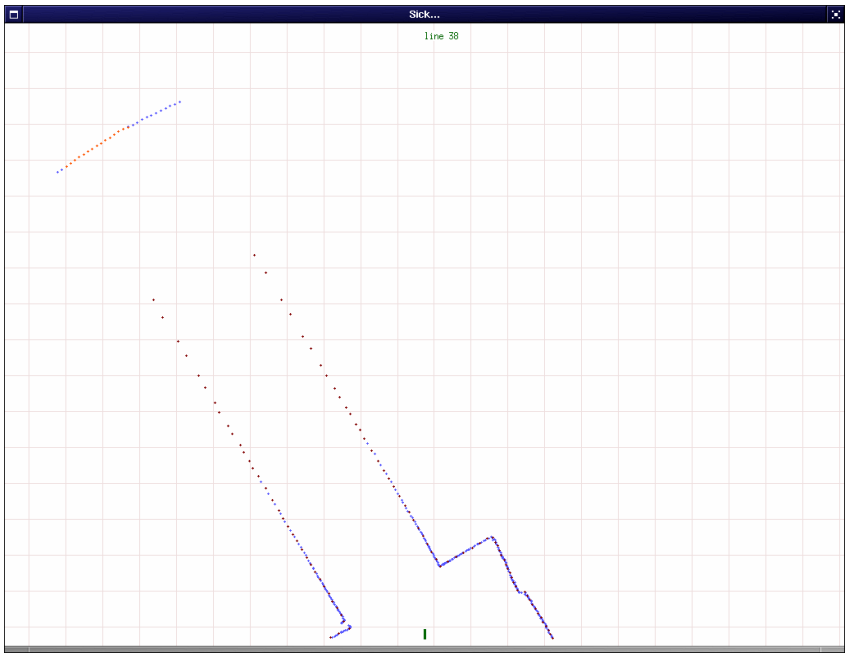


Figure B-3 : Direction la plus lointaine.
On extrait tout d'abord les mesures voisines correspondant aux plus grandes distances mesurées (points en orange). La « direction la plus lointaine » marque le milieu de cette région.

Le comportement précédent faisait osciller le robot sur la ligne du milieu du couloir. On peut le résumer par la stratégie suivante : « longer les murs ». Mais, on peut en définir une autre, tout aussi simple : « suivre la direction la plus lointaine », qui va correspondre à l'axe du couloir. Le robot va donc chercher à détecter l'angle pour lequel le capteur laser renvoie la plus grande valeur. Comme le capteur est limité en mesure à 8 mètres, on va avoir de nombreux angles correspondant à des mesures maximales. De plus, il peut arriver qu'une mesure soit absurde, et que le capteur renvoie pour un angle, une distance de 8m. On considère donc la région de points voisins qui sont détectés à la plus grande distance parmi les 360 mesures faites au temps courant. On appellera « direction la plus lointaine » l'angle correspondant à la mesure du milieu de cette région (cf. Figure B-3). Le comportement est très simple une fois qu'on considère cette mesure. Il suffit que le robot essaie de la maintenir à 0. Si on note θ_{loin} cet angle, la commande s'écrit :

$$\theta_{loin} = 0$$

b) Spécification.

- **Variables :**

Vrot : vitesse de rotation du robot.
θ_{loin} : direction la plus lointaine.

- **Décomposition :**

$P(Vrot, \theta_{loin}) = P(\theta_{loin}) \times P(Vrot \mid \theta_{loin})$.

- **Forme paramétrique:**

On représente $P(Vrot \mid \theta_{loin})$ par une gaussienne centré sur une vitesse de rotation proportionnelle à θ_{loin}:

$$\mathbf{P(Vrot \mid \theta_{loin})} = G\,(m(\theta_{loin}), \sigma(\theta_{loin})).$$

c) Identification.

Il reste à spécifier la moyenne et la variance de $P(Vrot \mid \theta_{loin})$. D'après la loi de commande (1), le robot cherche à ramener θ_{loin} à zéro. Si θ_{loin} est positif et augmente, Vrot doit être négatif et diminuer pour compenser. On pose donc :

$$\mu(\theta_{loin}) = -\theta_{loin}.$$

Pour simplifier, on choisit (comme plus haut): $\sigma(\theta_{loin}) = 1$.

d) Question.

La question à poser est :

$$P(Vrot \mid \theta_{loin}).$$

e) Evaluation du comportement.

Le comportement dépend finalement beaucoup moins de la configuration « en couloir » que le comportement précédent. On note donc que le robot se guide très bien, que ce soit le long d'un couloir le plus vide possible, dans un parcourt d'obstacle plus complexe, ou encore pour sortir d'une pièce.
Par contre, si le robot suit très facilement le couloir, il respecte moins bien la consigne initiale, qui était de ce placer ***au centre*** du couloir. Même si suivre la direction la plus lointaine le conduit finalement sur la ligne du milieu, ce replacement est lent. Après avoir évité un obstacle par exemple, le robot ne suit pas une trajectoire directe vers le centre du couloir, au contraire des comportements décrits plus haut. De plus, si le robot dévie légèrement de sa trajectoire, il ne corrigera pas immédiatement. La correction n'interviendra qu'une fois la divergence suffisamment importante pour modifier θ_{loin}. Enfin, le robot ne se place pas précisément au milieu du couloir, mais sur une bande de plus ou moins 20cm d'erreur. Dans les cas pathologiques, le comportement pose encore plus de problème. Quand le robot arrive en bout de couloir, la direction la plus lointaine ne se trouve plus forcément devant lui. Une porte ouverte sur le côté peut très bien devenir une direction intéressante pour le robot. Mais le comportement devient encore plus instable dans les couloirs dont un des murs est en verre. Dans ce cas, il arrive que le robot détecte une autre « direction lointaine » à travers la vitre.
Dans ce cas, il pivote brusquement pour la suivre. On pourrait corriger cette erreur par une intégration sur le temps précédent, qui favoriserait la direction correspondant à celle suivie au temps précédent. Malgré tout, si la distance la plus lointaine correspond à un angle proche de celui du couloir, le robot divergerait tout de même, suffisamment pour quitter complètement l'axe du couloir au point de rentrer en collision avec le mur. L'évitement d'obstacle ne permet pas d'éviter cette dernière erreur, car ce cas de figure intervient le plus souvent quand un des murs est en verre.
Au final, on voit que chercher à utiliser la longueur du couloir, à la manière de ce que ferait un humain, n'est pas la bonne solution dans notre cas. Certes le robot suit le couloir, mais pas de manière aussi constante que voulue.

Annexe C. Guidage dans une intersection.

C.1. Présentation.

On a vu que le comportement principal du robot était de suivre le couloir. Il faut maintenant lui permettre de repérer les intersections et, une fois qu'il en a détecté une, de choisir qu'elle direction prendre. Ce sont les deux parties que nous allons maintenant présenter.

C.2. Détection d'intersection.

C.2.1. Présentation.

Première chose à faire: détecter les intersections que le robot croise. Dans notre bâtiment, toutes les intersections sont en « croix », un couloir croisant à angle droit un autre couloir (cf. Figure 2-1-a). Il y a deux manières de détecter une intersection.
La première est la plus simple. Elle consiste à détecter sur la gauche et la droite du robot une brusque augmentation de la mesure renvoyée par les capteurs. C'est la méthode proposée dans {AYCARD98}. Le robot suit comme le notre un couloir, et détecte à l'aide de ces huit capteurs ultrasons les variations de la distance détectée. Puis, à l'aide d'une chaîne de Markov, le robot décide si il vient de passer devant une porte ouverte, devant une intersection en 'T', en 'X', ou bien si il arrive à la fin du couloir... Nous ne pouvons pas appliquer cette méthode dans notre cas. En effet, nous voulons que le robot reconnaisse l'intersection lorsqu'il la croise, voir avant si possible, mais certainement pas après l'avoir dépassée. De plus, le capteur de notre robot, un laser de grande précision et de grande portée, permet d'espérer ce résultat sans passer par une utilisation des chaînes de Markov. Plus simplement, nous cherchons une augmentation de la mesure capteur sur les côtés du robot. Quand cette augmentation atteint la limite de portée du capteur, sur un angle de 5°, nous considérerons que nous sommes au milieu d'une intersection. Evidement, une méthode aussi simple est sujette à des erreurs de détections. La restriction que nous apportons à notre modèle permet d'en éliminer la plupart:

- Nous considérons exclusivement des intersections en « X ». Il faut donc que les capteurs de droites et de gauches détectent chacun un passage de leur côté.
- nous considérons des intersections suffisamment larges pour que les mesures sur un angle de 5° consécutifs soient au maximum. Cela permet d'éliminer une grande partie des portes.
- nous exigeons une augmentation jusqu'à la valeur maximum du laser, soit une distance détectée de 8m. Cela élimine de nouveau une grande partie des portes, c'est à dire toutes celles donnant par exemple sur un bureau trop petit.

Malgré tout, cette détection n'est pas suffisante. Dans certains points précis du bâtiment, le robot détecte des zones respectant les consignes données plus haut, hors d'une intersection. Il faut donc trouver une autre propriété pour caractériser les intersections. On utilisera le mur du couloir que l'on croise. Il est facilement repérable sur les images du laser, comme on le voit sur la Figure C-1. Il est d'autant plus visible que le robot est centré dans le couloir qu'il longe, et d'autant plus facilement repérable que le robot est orienté dans l'axe du couloir. Cela justifie le choix des lois de commandes effectué dans la partie 4.1.
Le robot se déplace donc le long du couloir selon les commandes données en 4.1. Parallèlement, il détecte les augmentations dans les mesures renvoyées par le laser et les droites perpendiculaires à la direction du couloir. Pour cela, il utilise une nouvelle fois la transformée de Hough, de la même manière qu'en 4.1.7. On peut formaliser ce comportement par un programme bayésien.

Figure C-1 : Image laser d'une intersection.
Deux propriétés la caractérise: des mesures élevées sur les côtés du robot et une droite avec un fort score de probabilité perpendiculaire à la direction du couloir.

C.2.2. Spécification.

- **Variables:**

laser: ensemble des mesures laser.
couloir ∈ {V,F}: couloir détecté. Cette variable peut prendre deux valeurs: V si on croise un couloir, F si ce n'est pas le cas.
d, g ∈ {V,F}: espace détecté à droite, espace détecté à gauche.
Mur ∈ {V,F}: mur perpendiculaire. Cette variable prend la valeur V ssi un mur perpendiculaire est détecté avec une probabilité suffisamment forte, et à une distance inférieure à 1m.

- **Décomposition:**

$$P(\text{couloir}, \text{laser}, \text{mur}, d, g) = \left| \begin{array}{l} P(\text{laser}) \times P(\text{mur} \mid \text{laser}) \\ \times P(d \mid \text{laser}) \times P(g \mid \text{laser}) \\ \times P(\text{couloir} \mid \text{mur}, d, g) \end{array} \right.$$

On considère que les trois variables mur, g, et d sont indépendantes conditionnellement à laser. C'est l'hypothèse classique du modèle capteur. Cette hypothèse parait raisonnable car ces trois valeurs caractérisent des parties différentes des mesures laser (comme on peut le voir Figure C-1).
On considère de plus que la variable couloir est indépendante de la variable laser conditionnellement à la connaissance de mur, g et d. Cette hypothèse est beaucoup plus forte. Elle reprend la décision prise en 4.2.2.a. de ne considérer que ces trois caractéristiques des mesures laser pour prendre une décision. Evidemment, cela conduit à perdre énormément d'information, car la complexité des mesures renvoyées par le laser permettrait de prendre une décision beaucoup plus sûre (on en a la preuve en présentant une image laser à un humain. La réponse qu'il est capable de donner est toujours la bonne). Dans notre cas, nous espérons une réponse correcte. Mais surtout, la simplicité des variables choisies va permettre un temps de calcul très court.

- **Forme paramétrique:**

P([d = V] | laser): on considère les N premières mesures du laser. On en fait la somme, qu'on note S. Une ouverture est détectée avec certitude si S=Nx800 (la distance maximale renvoyée par le laser est 800cm). On choisit une fonction croissante selon S, valant 1 quand S vaut 800xN, et 0 quand S vaut 0. On choisit une gaussienne centrée en S/N = 800. Cela permet de considérer nulle la probabilité que d vaille V quand S/N est inférieur à 2m. La courbe est donnée Figure C-2.

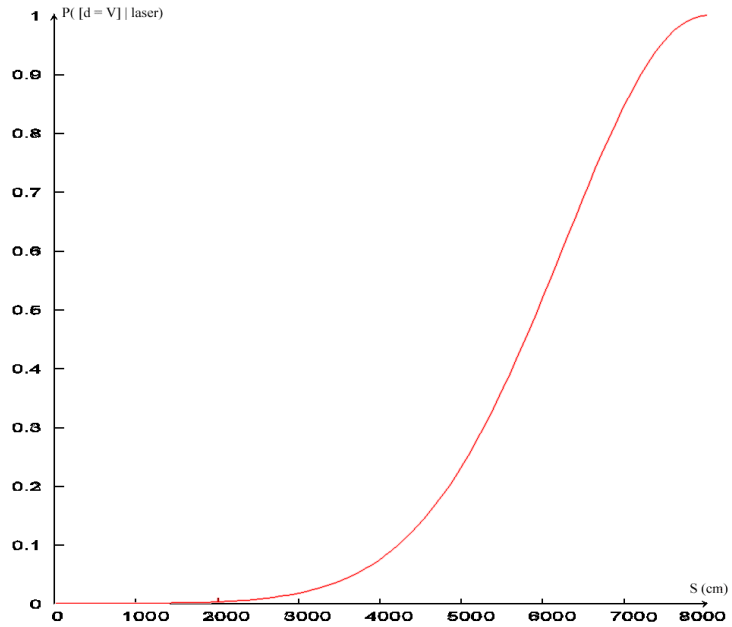


Figure C-2 : Forme paramétrique de P([d = V] | laser), représentée en fonction de S, la somme des N premières mesures renvoyées par le laser.

P([d = F] | laser) = 1 - P([d = V] | laser), par normalisation.
P([g = V] | laser): on choisit la même forme paramétrique que pour d, en calculant cette fois la somme S à partir des N dernières mesures. On prend de même :
P([g = F] | laser) = 1 - P([g = V] | laser).
P([mur = V] | laser): on utilise ici aussi une distance de Mahalanobis. La transformée de Hough renvoie une droite perpendiculaire à la direction du couloir. On la note :

$$\Delta = (\Phi, \text{dist}, \text{proba}),$$

avec **Φ** son angle, **dist** sa distance à l'origine, et **proba** la valeur de certitude qui lui est associé par la transformée de Hough. On considère que la probabilité de croiser un couloir est maximale quand ce vecteur vaut $\mu = (\pi/2, 80\text{cm}, 1)$. On pose donc :

$$P([\text{mur} = V] \mid \text{laser}) = f(\Phi, \text{dist}, \text{proba}) = \exp(-((\Delta - \mu)^T \cdot \sigma \cdot (\Delta - \mu)))$$

Par normalisation, on pose :

$$P([\text{mur} = F] \mid \text{laser}) = 1 - P([\text{mur} = V] \mid \text{laser}).$$

P([couloir = V] | mur, g, d): la détection d'une intersection dépend de trois caractéristiques qui peuvent avoir été détectées ou non dans l'image laser. On ne fixe pas de hiérarchie entre ces trois variables. La probabilité de détecter un couloir dépend donc seulement du nombre de propriétés vérifiées. Si on note N le nombre de variables parmi (mur, g, d) qui valent V, on peut écrire :

$$P([\text{couloir} = V] \mid \text{mur, g, d}) = f(N), \qquad \text{avec f strictement croissant.}$$

On utilisera $f(N) = \frac{N}{3}$. Cela revient à faire une fusion sur les trois informations mur, g et d.

C.2.3. Identification.

On a déjà choisit quasiment tous les paramètres des formes paramétriques dans la partie 4.2.2.b. Il ne nous reste plus qu'à fixer la variance de la gaussienne correspondant à P([mur = V] | laser). On prend σ diagonale, avec comme coefficients 1/Π, 1/500 et 1 :

$$\sigma = \begin{pmatrix} \frac{1}{\Pi} & 0 & 0 \\ 0 & \frac{1}{500} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

C.2.4. Question.

La question à poser est :

P(couloir | laser).

C.2.5. Evaluation du comportement.

Les trois variables choisies pour caractériser une intersection sont **suffisamment discriminantes** pour que la détection de couloir soit correcte, c'est-à-dire que le robot détecte des intersection dès qu'il en croise, et seulement quand il en croise. Quelques erreurs persistent malgré tout, en particulier quand le robot n'est pas au milieu du couloir au moment de croiser l'intersection. Quand le robot n'est pas au milieu, mais qu'il reste orienté parallèlement aux murs, la détection se fait correctement. Par contre, ce n'est plus le cas si le robot est désaxé. Dans ce cas, il détecte mal le mur perpendiculaire correspondant à la variable mur, et il ne détecte pas non plus les espaces libres sur ses côtés, correspondant aux variables d et g.
Un deuxième problème se pose, du au **limitation du modèle** que nous avons choisi. Nous ne considérons que des intersections en 'X'. Or, dans le bâtiment où se déplace le robot, certaines

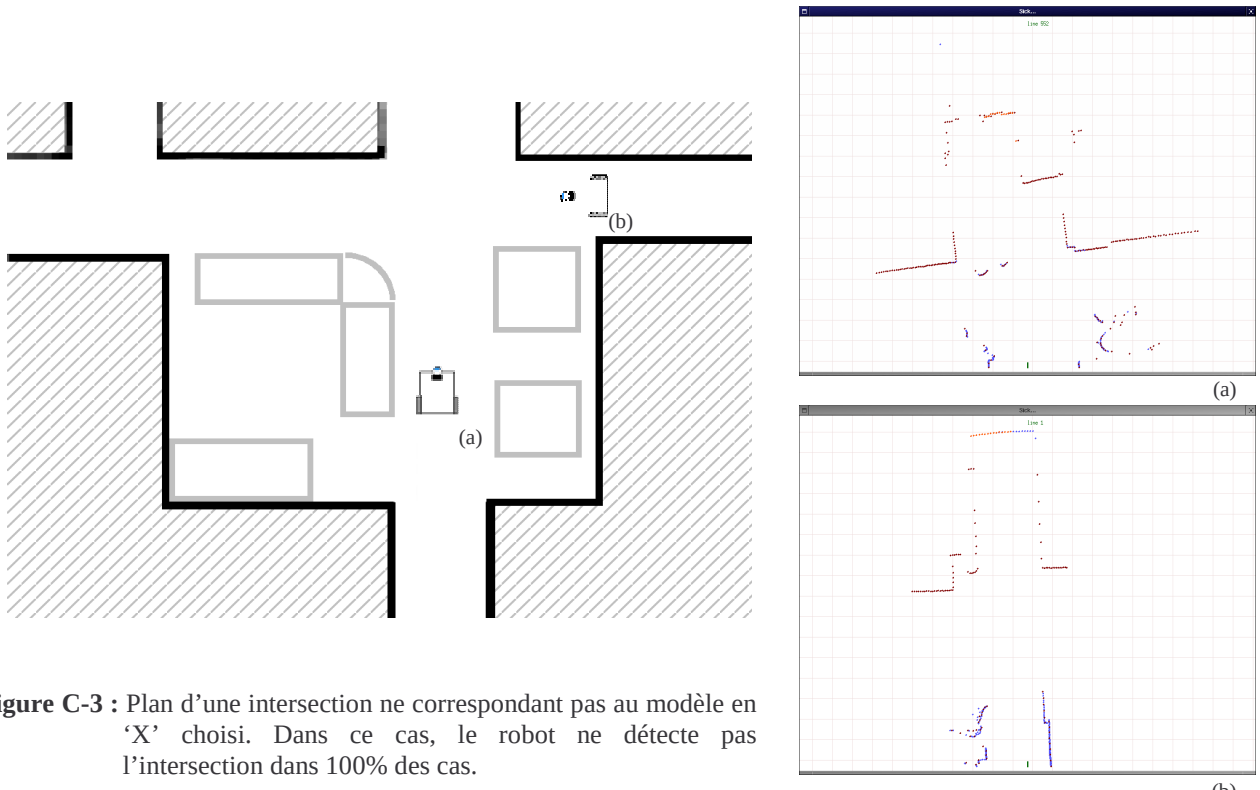


Figure C-3 : Plan d'une intersection ne correspondant pas au modèle en 'X' choisi. Dans ce cas, le robot ne détecte pas l'intersection dans 100% des cas.
(a) l'intersection est détectée sans problème, grâce au mur perpendiculaire.
(b) La probabilité de détecter l'intersection est plus faible.

intersections sont élargies, de sorte que **le modèle en 'X' n'est plus très cohérent**. Le plan d'une telle intersection est donné Figure C-3. Dans ces deux cas, il arrive que le robot ne considère pas le couloir qu'il croise comme une intersection. La probabilité P(couloir | laser) est plus élevée que lorsque le robot se situe entre deux murs droit, mais pas suffisamment pour que la réponse à la question bayésienne soit **V**. Or, nous ne voulons surtout pas que le robot rate une intersection, c'est-à-dire qu'il ne détecte pas l'intersection qu'il est en train de croiser. Il est par contre admissible qu'il voit une intersection alors qu'il n'y en a pas.

C.2.6. Confirmation d'intersection.

Pour cela, nous décidons qu'une intersection est probable dès que la probabilité P([couloir = V] | laser) dépasse un certain seuil (30%). Ce seuil a été fixé empiriquement. A partir de 30%, le robot ne rate aucune intersection. Par contre, il peut confondre deux portes ouvertes avec une intersection (bien que cela soit rare).
Pour corriger cette dernière erreur, le robot s'arrête quand la probabilité est assez forte. Il tourne alors de 90° a gauche, puis à droite. A chaque fois, il prend une image laser de ce qu'il voit. Dans les deux cas, il essaye de faire correspondre ces images au modèle de couloir présenté dans le chapitre 4. On considère que le robot est au milieu d'une intersection si la somme des certitudes de ces deux modèles est assez élevée.

C.3. Choix de la direction à prendre.

C.3.1. Présentation.

On considère maintenant que le robot se trouve au milieu d'une intersection. Quatre directions s'offrent donc à lui. Nous avons expliqué dans la partie 3 qu'il y avait de nombreuses manières de guider le robot une fois qu'il se trouve localisé aussi précisément. Les méthodes classiques utilisent un graphe topologique, dont les noeuds sont les intersections et les arêtes les couloirs. Les noeuds sont indexés par des caractéristiques de l'environnement, ce qui permet au robot de se localiser d'une manière globale. Par exemple, dans {BAILEY01}, les auteurs utilisent des primitives géométriques pour caractériser leur graphe. Dans {FRANZ98}, le robot se localise en comparant les images prises avec sa caméra panoramique, aux images qu'il a déjà vues. On a évoqué dans la partie 2.4 la possibilité de guider notre robot grâce à de telles méthodes.
Dans notre cas, nous ne cherchons pas à localiser le robot de manière globale, mais simplement à répondre à deux questions :

- suis-je suffisamment proche de la base pour changer de stratégie ?
- si non, lequel des quatre couloirs dois-je choisir pour m'en rapprocher ?

Nous avons vu dans la partie 3 que le robot marquait le couloir qu'il avait emprunté lors de son trajet aller par des boulons rouges. Il **suffit donc que le robot compte les boulons** qu'il voit grâce à la caméra dans chaque couloir. Si un des couloirs contient plus de cinq boulons, le robot peut considérer qu'il est arrivé suffisamment près de la base pour changer de stratégie. Sinon, il recommence à suivre le couloir qui contenait le plus de boulons.
Nous ne cherchons pas ici à utiliser de manière efficace la caméra. Le but de cette brique de base n'est pas de proposer une formulation bayésienne d'un problème ou de découvrir de nouvelle utilisation de la vision en robotique. Nous désirons simplement obtenir une information simple (combien de boulons rouges dans cette image ?). Ce programme à comme but de proposer un pré traitement rapide à mettre en place pour transformer les flots d'information en provenance de la caméra.

C.3.2. Comptage des boulons.

Le robot prend une image du sol de chacun des quatre couloirs. Pour chaque image prise, on désire calculer le nombre de zones rouges connexes détectées. L'algorithme se passe en deux parties. On crée tout d'abord une image de probabilité des boulons. Les boulons se repérés par leur couleur. Puis on regroupe les pixels de couleurs en un nombre minimal de zones de connexe. Chacune de ces zones correspondra à un boulon à condition de faire l'hypothèse qu'ils ne sont pas suffisamment proches pour être confondus. Au final, on considèrera qu'on a autant de boulons que de zones de couleurs connexes.

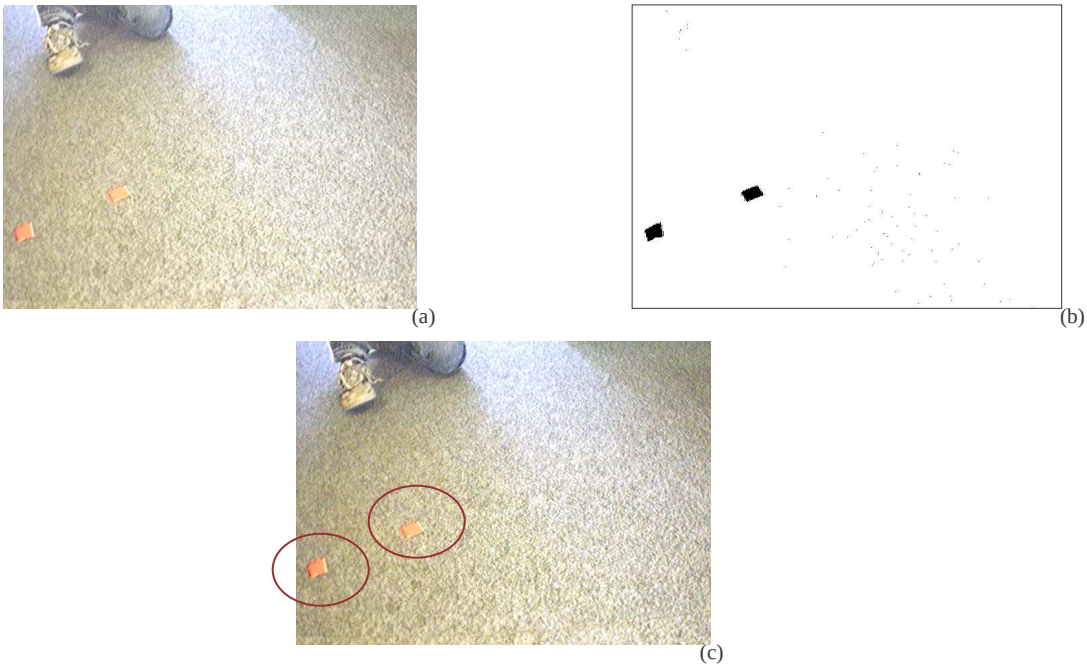


Figure C-4 : (a) Image prise par la caméra.
(b) Image de probabilité correspondante.
(c) Regroupements éffféctués à la fin de l'algorithme: le robot voit trois boulons.

C.3.3. Image de probabilité.

Une formalisation bayésienne de la construction d'une image de probabilité est donnée dans {COUE00}. L'auteur utilise pour la construire un histogramme de l'objet, que le programme a appris avant l'exécution. Pour notre part, nous n'allons pas utiliser d'histogramme de couleur. En effet, les boulons paraissent très simple à détecter: les boulons sont rouges vifs sur fond gris.
L'image obtenue de la caméra est au format RGB. Nous transposons les pixels dans l'espace de couleur HSV. Nous obtenons la probabilité qu'un pixel corresponde à un boulon par une simple gaussienne sur la couleur rouge pure, avec une covariance diagonale. On obtient l'image présentée sur la Figure C-4-b.

C.3.4. Regroupement des pixels connexes.

On obtient un certain nombre de pixel ayant une probabilité élevée d'appartenir à des boulons. Une première solution serait simplement de compter le nombre de pixels de la bonne couleur. Cela donne une évaluation du nombre de boulons détectés. Il suffirait alors que le robot suive la direction où il détecte le plus de pixels rouges. Mais cette stratégie donnerait plus d'importance aux boulons proches du robot, qui sont vus plus grands (donc avec plus de pixels) par la caméra. La marge d'erreur sur la position du robot au milieu de l'intersection devrait alors être réduite.

Pour plus de souplesse, nous choisissons donc d'étudier le nombre de parties connexes. Le but est de regrouper tous les pixels dont la probabilité est assez élevée par voisinage. On veut obtenir une partition de l'ensemble des pixels rouges telle que deux pixels dont la distance est inférieure à un seuil fixé au départ soient dans le même sous ensemble. L'algorithme est détaillé dans l'annexe A {!!!}. Après son exécution, on obtient N sous parties C_1 .. C_N, de taille T_1 .. T_N. Le nombre de boulons détectés par l'algorithme vaut :

$$Nb = Card \{ C_i \text{ tq } T_i > SEUIL \}.$$

Le seuil a été fixé empiriquement à 200 pixels.

C.3.5. Evaluation du comportement.

On utilise ici la caméra comme un détecteur très spécifique. On y perd beaucoup d'information. La caméra est finalement utilisée uniquement pour compléter le capteur laser. On ne voulait rien de plus. Grâce à cette partie du programme, on a finalement l'équivalent du détecteur d'alcool utilisé pour guider le robot dans {MENG93}.
Il faut noter que la détection des points de couleurs n'est pas excellente. Ce n'était pas le but de ce travail, aussi n'avons nous pas cherché à développer de notre côté des techniques déjà bien connu des équipes travaillant sur la vision.
Le détecteur est suffisamment peu sensible pour donner les réponses correctes dans l'environnement dans lequel travaille le robot. Mais il reste sensible aux grosses variations de lumière. En particulier, les tests sont généralement effectués le soir, à la lumière artificielle, quand les locaux sont vides. Pendant la journée, le robot à parfois du mal à obtenir le compte correct de boulon.
Finalement, il faut préciser que le distributeur de boulons n'est pour l'instant qu'un prototype. Le robot n'est pas capable pour l'instant de déposer des boulons par lui-même. Nous utilisons à la place des petits bouts de papier rouge, d'environ 2x2cm, placés à la main.

Annexe D. Suivi de direction par amer visuel.

D.1. Présentation.

Le robot cherche à chaque intersection une réponse aux questions :

- suis-je suffisamment proche de la base pour changer de stratégie ?
- si non, lequel des quatre couloirs dois-je choisir pour m'en rapprocher ? (cf. Figure 3-1).

Le comportement que nous présentons ici est adopté quand le robot répond OUI à la première question. Le robot doit alors avancer jusqu'à être capable de repérer la base visuellement. La base en question est la borne sur lequel le robot doit se brancher pour se recharger. Celle-ci doit être facilement visible, et facilement repérable, grâce notamment à une marque distinctive. Ici encore, notre but n'est pas d'utiliser de manière optimale les informations en provenance de la caméra.
Nous marquerons donc la cible d'une grosse tache de couleur rouge. Nous considérerons que seule la cible est de couleur rouge dans la pièce où se déplace le robot. Nous proposons ici de fusionner deux comportements. Le premier consiste à suivre l'amer visuel 'tache rouge'. C'est le comportement prioritaire, à partir du moment où le robot voit la base. Pour s'en rapprocher, le second comportement consiste à suivre la direction la plus lointaine. Nous reprenons ici le programme déjà présenté.

D.2. Suivi de l'amer visuel.

Nous présentons ici rapidement le travail effectué dans {COUE00}. Le but était de poursuivre une cible de couleur rouge avec un robot mobile doté d'une caméra pivotante. La cible est détectée à l'aide d'un histogramme de couleur.
L'idée principale est de faire pivoter la tête du robot (la caméra) pour conserver la cible au centre de l'image. Ensuite, il suffit de faire pivoter le corps du robot à la suite de la tête. Un prétraitement sur l'image de la caméra donne la position du centre de gravité de la cible rouge détectée. Pour notre part, nous ne considérons que la position horizontale de la cible (c'est à dire le 'Pan' de la caméra pan tilt). On peut le formaliser grâce au programme bayésien présenté ci-dessous.

- **Variables:**

Pos_Cible : coordonnée horizontale de la cible dans l'image.
D_Pan : déplacement de la caméra (on ne la commande pas en vitesse directement, mais en déplacement, c'est-à-dire par la valeur notée habituellement **dx**).
Pos_Pan : orientation de la caméra par rapport au corps du robot.
Vrot : vitesse de rotation du robot.

- **Décomposition:**

$$P(\text{Pos\_Cible}, \text{D\_Pan}, \text{Pos\_Pan}, \text{Vrot}) = \left| \begin{array}{l} P(\text{Pos\_Cible}) \cdot P(\text{D\_Pan} \mid \text{Pos\_Cible}) \\ \cdot P(\text{Pos\_Pan}) \cdot P(\text{Vrot} \mid \text{Pos\_Pan}) \end{array} \right.$$

Nous faisons ici une hypothèse très forte: nous supposons que la position de la caméra Pos_Pan ne dépend pas de son déplacement D_Pan.

- **Forme paramétrique:**

P(Pos_Cible), P(Pos_Pan) : nous n'avons pas d'a priori sur ces variables. Nous choisissons donc une distribution uniforme.
P(D_Pan | Pos_Cible) = $\mathcal{G}(\mu(\text{Pos\_Cible}), \sigma(\text{Pos\_Cible}))$. La position de la caméra dépend directement de la position de la cible.
P(Vrot | Pos_Pan) = $\mathcal{G}(\mu(\text{Pos\_Pan}), \sigma(\text{Pos\_Pan}))$. Là encore, nous décidons de la vitesse de rotation du corps directement en fonction de la position de la tête.

- **Identification.**

Toujours pour simplifier, nous fixons les variances des deux gaussiennes. Quand la cible est détectée à gauche, la caméra doit se déplacer sur la gauche. On choisit donc:

$$\mu\ (Pos\_Cible) = K_1\ .\ Pos\_Cible, \qquad K_1 > 0.$$

De même, quand la caméra s'est déplacé sur la droite, il faut que le robot se déplace aussi sur la droite. On choisit donc de la même manière:

$$\mu(Pos\_Pan) = K2\ .\ Pos\_Pan, \qquad K_1 > 0.$$

- **Question.**

La question à poser est :

$$P(Vrot, D\_Pan \mid Pos\_Pan, Pos\_cible).$$

- **Conclusion.**

Pour plus de détail, nous invitons le lecteur à se référer à {COUE00}. Pour résumer, le robot ainsi programmé était capable de suivre une cible mobile de la bonne couleur. Nous avons ici simplifié le programme à l'essentiel pour notre problème. En particulier, nous avons fixé la vitesse de translation du robot, ainsi que la position verticale de la caméra, sans tenir compte de la position verticale de la cible. Dans le programme originel, le robot décidait lui-même de sa vitesse de translation en fonction entre autre de la position Tilt de la caméra.
Par contre, le comportement devenait incohérent dans deux cas : quand la cible ***disparaissait*** du cadre de l'image, et quand une ***seconde cible***, c'est à dire une tache de couleur parasite apparaissait dans l'image.
Nous avons fait dans notre cas l'hypothèse que le seul objet rouge de l'environnement serait la base. Nous ne nous intéressons donc pas au second problème. Par contre, le robot ***ne voit pas la cible*** dès le départ. Nous devons donc prévoir une réponse au second problème.

D.3. Suivi de la direction la plus lointaine.

On rappelle simplement ici les variables et les questions détaillées dans l'annexe B :

- **Variables:**

$\boldsymbol{\theta_{loin}}$: direction de la mesure la plus lointaine renvoyée par le capteur laser.
Vrot : vitesse de rotation du robot.

- **Question:**

$$P(Vrot \mid \theta_{loin}).$$

D.4. Fusion des deux comportements.

Un pré traitement nous donne la position du centre de gravité des pixels rouges dans l'image. Le problème est que ce prétraitement renvoie toujours une valeur, même quand aucune cible raisonnable n'est détectée. Nous cherchons donc un critère qui nous permette de juger la validité du suivi d'amer. Nous utiliserons pour ceci la taille (en nombre de pixel) de la cible détectée. Le programme final est simple: il s'agit d'une fusion entre les comportements (D.2) et (D.3), conditionnée à la taille de la cible. On pourra alors fusionner le résultat avec un évitement d'obstacle, comme présenté dans le chapitre 5.

D.4.1. Prétraitement.

De l'image enregistrée par la caméra, nous extrayons l'image de probabilité que chaque pixel appartienne à la cible. Pour chaque pixel (x,y) de l'image, on note p la probabilité qu'il appartienne à la cible.

$$p(x,y) = P((x,y) \in \text{Cible} \mid \text{Couleur}(x,y)).$$

Puis, nous calculons la moyenne et la variance de l'ensemble des pixels ayant une probabilité supérieure à un seuil fixé au départ. Nous obtenons alors un vecteur de dimension 2 représentant le centre de la tâche de couleur, et une matrice de dimension 2x2, représentant l'étendu de cette tâche.

$$\vec{\mu} = \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix} = \frac{1}{N} \begin{pmatrix} \sum_{X,Y} p(X,Y) \cdot X \\ \sum_{X,Y} p(X,Y) \cdot Y \end{pmatrix}$$

$$\sigma = \begin{pmatrix} \sigma_{XX} & \sigma_{XY} \\ \sigma_{XY} & \sigma_{YY} \end{pmatrix} = \frac{1}{N} \begin{pmatrix} \sum_{X,Y} p(X,Y) \cdot (X-\mu_X)^2 & \sum_{X,Y} p(X,Y) \cdot (X-\mu_X) \cdot (Y-\mu_Y) \\ \sum_{X,Y} p(X,Y) \cdot (X-\mu_X) \cdot (Y-\mu_Y) & \sum_{X,Y} p(X,Y) \cdot (Y-\mu_Y)^2 \end{pmatrix}$$

On peut alors approximer l'image de probabilité par une gaussienne. Pour calculer une estimation de la taille de la tache, nous nous intéressons au nombre de pixel compris dans la gaussienne, de probabilité p(x,y) suffisamment élevée.

$$\text{TAILLE} = \text{Card}\ \{\ (x,y) \mid p(x,y)\ .\ \mathcal{G}(x,y) > \text{SEUIL}\ \}.$$

D.4.2. Spécifications.

- **Variables :**

Vrot : vitesse de rotation du robot.
Pos_cible : position de la cible par rapport au centre de l'image.
taille : taille en nombre de pixel de la cible détectée.
D_Pan : déplacement de la caméra.
Pos_pan : position de la caméra par rapport au corps du robot.
θ_{loin} : direction de la mesure laser la plus élevée.
H ∈ (AMER, DIR) : comportement adopté. H vaut **AMER** si le robot suit la cible qu'il détecte, **DIR** si il se dirige vers la plus grande mesure capteur.

- **Décomposition :**

$$P(Vrot, Pos_{cible}, taille, D_{Pan}, Pos_{Pan}, \theta_{loin}, H) = \left| \begin{array}{l} P(Pos_{cible}) \cdot P(taille) \\ \cdot\ P(Pos_{Pan}) \cdot P(\theta_{loin}) \\ \cdot\ P(H \mid \text{taille}) \cdot \text{P}(\text{D}_{\text{Pan}} \mid \text{Pos}_{\text{cible}}, H) \\ \cdot\ \text{P}(\text{Vrot} \mid Pos_{Pan}, \theta_{loin}, H) \end{array} \right.$$

- **Forme paramétrique :**

On fusionne les deux comportements précédents. Il suffit donc de fixer H puis de reprendre les distributions précédentes:
P(D_Pan | Pos_Pan, [H = AMER]) = P(D_Pan | Pos_Pan)
P(D_Pan | Pos_Pan, [H = DIR]) = uniforme. Cette distribution permet de faire bouger la caméra aléatoirement quand le robot cherche sa cible, ce qui augment la probabilité de la trouver. Comme

le contrôle en position de la caméra possède une buté, il faudra ensuite réaliser une vérification sur la vitesse appliquée en fonction de la position courante. Ceci n'a pas besoin d'être spécifié en bayésien.

$P(Vrot \mid \theta_{loin}, Pos\_Pan, [H = AMER]) = P(Vrot \mid Pos\_Pan)$.

$P(Vrot \mid \theta_{loin}, Pos\_Pan, [H = DIR]) = P(Vrot \mid \theta_{loin})$.

Enfin, il nous reste à spécifier les a priori sur H. On désire suivre l'amer visuel si la cible est détectée de manière sûre. On modélise cet a priori par une sigmoïde.

$$P([H = AMER] \mid taille) = \frac{1}{1 + \exp((A - taille) \times B)}.$$

$$P([H = DIR] \mid taille) = 1 - P([H = AMER] \mid taille).$$

D.4.3. Identification.

On reprend les mêmes paramètres que précédemment. Il nous reste à fixer les paramètres de la sigmoïde. A est le seuil de pixel nécessaire pour suivre la cible. En particulier, si taille vaut A :

$$P([H = AMER] \mid taille) = P([H = AMER] \mid taille) = ½.$$

On prend A = 500 pixels. B correspond à la vitesse de croissance de la courbe. On prend B=1/100.

D.4.4. Question.

On pose la question :

$$P(Vrot, D\_Pan \mid Pos\_Pan, Pos\_Cible, taille).$$

D.4.5. Evaluation du comportement.

Faute de temps, ce comportement n'a pas pu être testé. Dans les tests du comportement global, nous avons considéré que le robot était arrivé à la base quand il était capable de la voir. Au vu des résultats obtenus dans {COUE00}, il nous semble que le comportement général serait correct. Le robot pourra nous semble-t-il se rapprocher jusqu'à une faible distance de la base.

Le but de ce comportement était d'amener le robot à se brancher sur la base. Il faut pour cela qu'il s'en rapproche jusqu'à être en contact avec elle. De plus il faut que la phase finale de l'approche se fasse perpendiculairement à la base pour que le branchement soit correct. En utilisant une seule tache de couleur, il semble peu probable que le robot réussisse à effectuer sa manœuvre. Pour cela, il faudrait apporter une information supplémentaire, qui lui permettrait de déduire son angle d'approche. A partir du moment où il s'en approche suffisamment il pourrait par exemple se servir du capteur laser pour connaître l'angle de la base avec sa trajectoire courante.

Annexe E. « Je suis perdu ! »

E.1. Présentation.

Ce dernier comportement est le plus simple. Il s'agit simplement d'offrir au robot une possibilité de se guider si le modèle du couloir qu'il a reconstruit (comme expliqué dans le chapitre 4) devient trop incertain. Ceci arrive essentiellement quand le robot atteint la fin du couloir.
La première alternative que le robot doit appliquer consiste à utiliser le comportement décrit en 4.1.7. Le robot se met à suivre la direction la plus lointaine qu'il peut trouver. Dans le cas d'un couloir, on l'a vu, cela correspondra à suivre le couloir lui-même. Quand le robot arrive à la fin du couloir, il va continuer à le suivre jusqu'à trouver une autre issue, par exemple une porte ouverte sur le côté. Ce comportement est très intéressant car il contient lui aussi une variable qui peut servir de diagnostique. En effet, ce comportement est raisonnable si la distance la plus lointaine détectée est grande. Dans ce cas, on peut considérer qu'avancer dans cette direction conduira à une position intéressante. Par contre, si la distance se réduit, cela signifie qu'on arrive dans un cul de sac. En dessous d'un certain seuil, il parait clair qu'aucune issue n'apparaîtra.
La seconde alternative intervient quand la première solution devient trop incertaine. Cela signifie généralement que le robot est arrivé dans une impasse. La stratégie la plus raisonnable est alors de tourner sur lui-même à la recherche d'une issue, c'est à dire d'une augmentation de la distance la plus lointaine jusqu'au dessus du seuil critique.

E.2. Spécifications.

- **Variables:**

Vrot: vitesse de rotation du robot.
Vtrans: vitesse de translation du robot. Jusqu'à présent, on a considéré uniquement la vitesse de rotation, laissant la vitesse de translation constante. Ici, le robot peut se mettre à tourner sur lui même. Pour cela, il faut annuler la vitesse de translation. On la considère donc comme une variable.
θ_{loin}: direction de la mesure la plus lointaine (cf. 4.1.7.).
$DIST_{loin}$: valeur de la mesure la plus grande renvoyée par le laser, correspondante à θ_{loin}.

- **Décomposition:**

$P(Vrot, Vtrans, \theta_{loin}, DIST_{loin}) = P(\theta_{loin}) \times P(DIST_{loin}) \times P(Vrot \mid \theta_{loin}) \times P(Vtrans \mid DIST_{loin})$.

- **Forme paramétrique:**

$P(\theta_{loin})$, $P(DIST_{loin})$: on n'a pas d'a priori sur ces variables. On choisit donc une distribution uniforme.
$P(Vrot \mid \theta_{loin})$: on choisit comme en 4.1.7. de représenter cette distribution par une gaussienne centré sur une vitesse de rotation proportionnelle à θ_{loin}:

$$P(Vrot \mid \theta_{loin}) = \mathcal{G}(\mu(\theta_{loin}), \sigma(\theta_{loin})).$$

$P(Vtrans \mid DIST_{loin})$: La vitesse de translation doit être élevé quand le comportement est sûre, c'est à dire quand $DIST_{loin}$ est grand (proche de 800), et nulle quand le comportement devient incohérent, c'est à dire quand $DIST_{loin}$ est petit (inférieur à 1m). On choisit de représenter la distribution par une gaussienne centrée autour d'une vitesse dépendant de la distance:

$$P(Vtrans \mid DIST_{loin}) = \mathcal{G}(\mu(DIST_{loin}), \sigma(DIST_{loin})).$$

E.3. Identification.

Pour simplifier le problème, on prend les deux variances constantes égales à un. Il nous reste à fixer les deux moyennes.Comme dans la partie 4.1.7, on prend :

$$\mu(\theta_{loin}) = -\theta_{loin}.$$

La vitesse de rotation doit être nulle quand $DIST_{loin}$ est petit, et augmenter rapidement une fois passé le seuil. On choisit donc de le représenter par une sigmoïde:

$$\mu(DIST_{loin}) = \frac{VTRANS}{1+\exp(\frac{1}{K}\cdot(SEUIL - DIST_{loin})}$$

Le seuil correspond à la valeur où la vitesse bascule de 0 à une valeur plus grande. On prend SEUIL = 100 = 1m. K fixe la vitesse de décroissance de la courbe. On prend K = 10 en pratique. VTRANS enfin donne la vitesse 'de croisière' du robot, c'est à dire celle qui est utilisée habituellement pour faire avancer le robot. On choisit VTRANS=1m/s. C'est la vitesse utilisée habituellement.

E.4. Question.

La question à poser est :

$$P(Vrot, Vtrans \mid \theta_{loin}, DIST_{loin}).$$

E.5. Evaluation du comportement.

Ce comportement est très simple. Il reprend de plus les résultats du comportement étudié dans la partie 4.1.7. Le robot exécute exactement ce que nous attendions de lui.

Pour bien faire comprendre l'utilité de ce comportement, le plus simple est de présenter un cas où il est utilisé par le robot pour se sortir d'une situation dans laquelle la stratégie « suivre le couloir » ne fonctionne pas. Nous nous plaçons à la fin d'un des couloirs desservant les ailes du bâtiment. Le

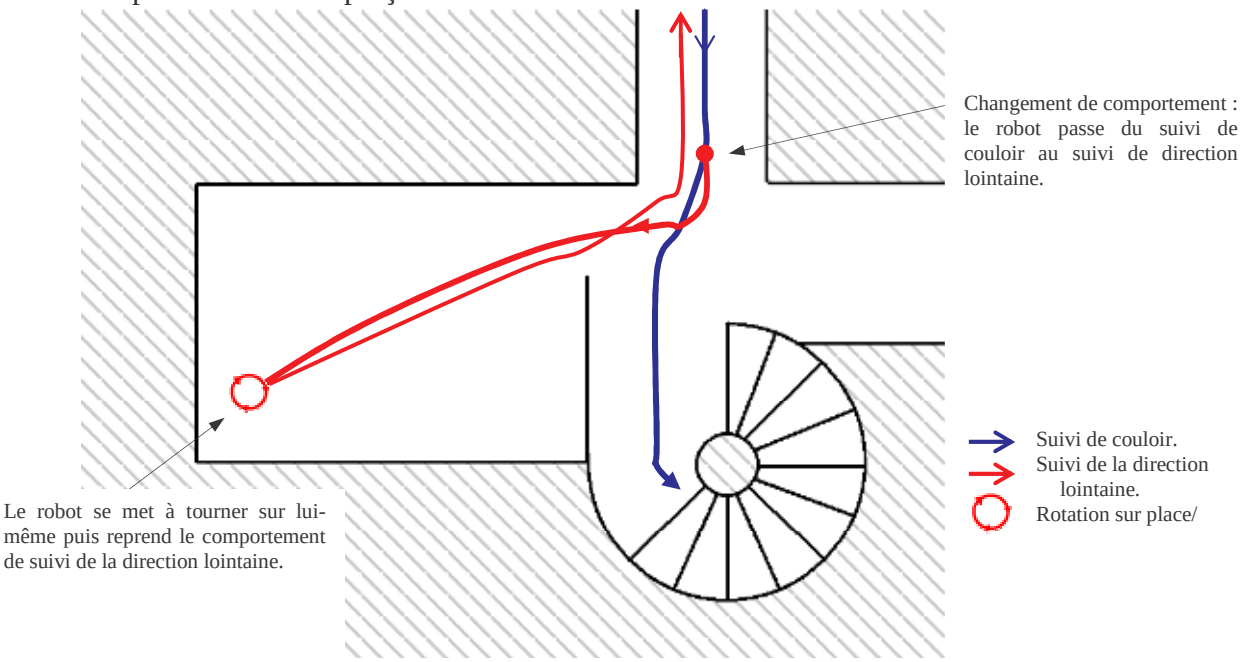


Figure E-1 : Fin d'un couloir dans le bâtiment de l'INRIA. Dans cette configuration, suivre le couloir conduit à aller se bloquer sous l'escalier. Par contre, en adoptant le comportement alternatif, le robot entre dans la pièce, puis fait demi-tour est rejoint le couloir. Sa trajectoire est présentée en rouge sur la figure.

couloir se termine par un réduit qui s'enfonce en se rétrécissant sous un escalier en colimaçon et par une pièce de 5m de long qui s'ouvre sur le côté droit. La carte des lieux est donnée Figure E-1.
Comme on l'a dit dans la partie 4.1, le comportement « suivre le couloir », conduit le robot à continuer le plus possible tout droit. Cela va l'amener à considérer le réduit situé devant lui comme un petit couloir. Au final, le robot va se bloquer sous l'escalier. Il faut préciser que l'indice de certitude du modèle de couloir reconstruit devient très faible avant que le robot s'engage dans le couloir (à partir du point rouge sur la Figure E-1).
En utilisant le comportement que nous présentons ici, le robot s'engage dans la pièce sur sa droite. La pièce fait 5m de long, il s'avance donc largement, jusqu'à atteindre le coin. Il se trouve alors vraiment à la fin de l'impasse. La mesure la plus grande qu'il détecte vaut alors moins de 1m50. Il ralenti jusqu'à s'arrêter, et tourne sur lui même. Quand il a suffisamment tourné, le laser se met à détecter la porte par lequel il est entré, et le couloir derrière lui. Sa vitesse de translation augment alors, et il se remet en marche. Il sort finalement de la pièce et rejoint le couloir.

Annexe F. Statistiques de réussite des tests.

Nombre de retour à la base	**15**
Succès	**7** (46 %)
Retours jusqu'à la dernière intersection	**15** (100 %)
Distance totale parcourue	2850m
Intersections traversées	**57**
Intersections détectées	**72** (erreur 26 %)
Erreurs de choix de direction	**7**
Utilisations du comportement « je suis perdu »	**18**

Figure F-1 : Statistiques de réussite des tests effectués dans le bâtiment de l'INRIA.
Le robot est amené dans un des bureaux de l'aile E 5à gauche au centre sur la figure 3.1. Des marques sont placé à toutes les intersections, d'après les résultats obtenus grâce au simulateur (carte obtenue après 100 parcours d'une longueur moyenne de 5 intersections).

Les erreurs principales viennent du traitement d'image. L'image de probabilité obtenue lors du comportement « guidage dans une intersection » est de très mauvaise qualité. L'étude des parties connexes a de plus été traité trop rapidement pour garantir un bon taux de reconnaissance. Ces erreurs expliquent aussi le faible taux de succès, alors que le robot arrive sans problème jusqu'à la dernière intersection.
Les tests valident donc l'implémentation de la hiérarchie de comportements, et confirme en même que certains comportements sont trop simples pour garantir une implémentation sans faille.