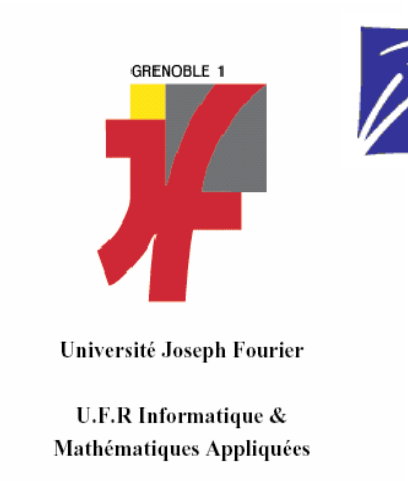
Recherche et poursuite d'une cible mobile par un robot

Jerome de Saint Jean



I. M. A. G.
DEA IVR :
IMAGERIE, VISION ET ROBOTIQUE

Projet présenté par :

Jérôme DE SAINT JEAN

Recherche et poursuite d'une cible mobile par un robot

Effectué au laboratoire : GRAVIR
Equipe LAPLACE - Tuteur : Olivier Aycard

Date : 18 juin 2003

Jury : O. AYCARD
S. COQUILLARD
A. LUX
R. SIEGWART

Sommaire

Résumé

Mots-clés : Planification, Processus Décisionnel de Markov, Localisation markovienne, Robotique, Comportement animal.

Un robot mobile évoluant dans un environnement réel doit faire face à de nombreuses incertitudes. Les actions qu'il exécute n'ont pas toujours l'effet escompté, et ses capteurs lui renvoient des données souvent bruitées. Les Processus Décisionnels de Markov (PDM), du fait de leur adaptation à la prise en compte d'incertitudes, sont étudiés depuis une dizaine d'années dans la communauté « Intelligence Artificielle ». Les plans obtenus sont donc beaucoup plus robustes aux incertitudes que ceux définis par des méthodes déterministes.

C'est dans ce cadre que s'inscrit ce stage. Nous nous sommes intéressés à une étude théorique des Processus Décisionnels de Markov et à leur application à la robotique. Nous avons pu montrer les limites et les avantages de cette approche. Nous avons ainsi pu appliquer les PDM à un problème complexe : la recherche d'un but mobile par un robot mobile. Sachant que leurs positions initiales respectives sont inconnues. Ce problème s'apparente au comportement animal d'un prédateur chassant sa proie.

Par ailleurs, de nombreuses perspectives se présentent, nous pouvons citer la possibilité de choisir la stratégie de localisation lors d'une exécution, la considération de plusieurs cibles ou la prise en compte des capteurs situés à l'arrière du robot.

Les premiers résultats expérimentaux montrent clairement que l'application de cette méthode lors d'une phase d'exécution réelle demeure acceptable. La localisation Markovienne donne de bons résultats. Cependant, ils sont fortement tributaires des différents réglages des paramètres des modèles. Enfin, la recherche et la poursuite d'une cible fournissent des résultats corrects. La localisation est très efficace lors de la recherche de la cible mais montre une faiblesse lors de la poursuite. Un couplage à un système vidéo semble le plus adéquat pour cette phase.

Remerciements

Je tiens à remercier mon tuteur, Olivier Aycard pour ses conseils, pour sa disponibilité tout au long de ce stage et de m'avoir permis d'effectuer un travail sur un sujet très intéressant où j'ai pu bénéficier d'une certaine marge de manœuvre.

Je remercie également le laboratoire GRAVIR de m'avoir accueilli, ainsi que les membres de l'équipe LAPLACE.

Mes remerciements vont aussi aux différents membres de mon jury :

- Madame Sabine Coquillard,
- Monsieur Augustin Lux,
- Monsieur Roland Siegwart

1. Introduction

1.1. Sujet de recherche

La planification est une des nombreuses branches de l'intelligence artificielle. Elle s'intéresse à trouver une séquence d'actions permettant de passer d'un état initial à un état final. Les premiers travaux dans ce domaine considèrent les effets des actions comme déterministes, c'est-à-dire que chaque action est toujours exécutée de la même façon et a toujours les mêmes effets.

La robotique moderne utilise la planification pour pouvoir rendre ses robots « intelligents ». Pour cela, le robot doit posséder quelques capacités humaines : perception de l'environnement, acquisition de connaissances, prise de décision … Ainsi, les robots de première génération qui étaient des automates, étaient uniquement destinés à répéter une suite de mouvements au sein d'un environnement statique et n'avaient que peu besoin d' « intelligence ». Il n'en est pas de même des générations futures. En effet, ces besoins se sont accrus au fur et à mesure de l'évolution de la robotique. Les robots actuels, dits de troisième génération, se doivent d'être mobiles, autonomes, de planifier leurs actions et de réagir aux stimuli de l'environnement.

La robotique mobile constitue un nouveau champ d'application très intéressant pour les techniques de planification du fait que nous ne pouvons pas partir du principe que « tout se passera comme prévu ». En effet, un robot mobile exécutant une suite d'actions le menant d'un point initial à un but dans un environnement réel est soumis à de nombreuses incertitudes : le glissement de ses roues, la rencontre d'obstacles mobiles, imprévus ou encore les erreurs dues au bruitage de ses capteurs. Ces incertitudes, même si elles ne sont pas significatives à court terme le seront à long terme. Les imprécisions résultantes introduisent un problème de localisation, le robot s'éloigne progressivement de la position prévue lors de la planification et la suite d'actions envisagée pour joindre le but sera faussée.

C'est dans ce cadre que s'inscrit l'activité de recherche de l'équipe LAPLACE du laboratoire GRAVIR.

Au début des années 90, des études novatrices dans le cadre de l'intelligence artificielle sont apparues. Elles concernent l'utilisation d'une certaine classe de modèles stochastiques très adaptés à la prise en compte d'incertitudes : les Processus Décisionnels de Markov (PDM). Ils sont une classe particulière des modèles stochastiques.

C'est pourquoi, lors de ce stage, nous nous sommes intéressés à l'application des Processus Décisionnels de Markov à la robotique mobile.

1.2. Plan du rapport

La suite de ce rapport est organisée comme suit :

- Dans le chapitre 2, nous présenterons les modèles stochastiques que nous allons utiliser. Nous définirons les différentes politiques et détaillerons les algorithmes associés.
- Le chapitre 3 est consacré à la mise en oeuvre d'un PDM dans le cadre de la robotique mobile. Nous montrerons tout d'abord la structure de PDM que nous avons choisie (états représentant l'environnement, actions, traitement des obstacles, etc.). Puis, nous illustrerons sur des exemples le type de plans (appelés politiques dans la littérature consacrée aux PDM), que nous obtenons en utilisant notre modèle. Puis nous justifierons notre choix de l'algorithme de résolution, avant de montrer comment il peut être facilement accéléré grâce à une initialisation astucieuse. Nous présenterons alors les résultats obtenus.
- Dans le chapitre 4, nous poursuivrons par une partie consacrée à la présentation du robot. Nous définissons ses caractéristiques et ses capteurs. Nous décrivons la fonction d'observation choisie et son adaptation à notre problème. Puis, nous exposons les propriétés d'un environnement réel. Ensuite, nous explicitons le déplacement du robot : ses mouvements, les cas litigieux et l'évitement d'obstacles. Enfin, nous exposons l'utilisation réelle du robot.
- Dans le chapitre 5, nous nous intéressons à la résolution d'un problème complexe : le robot doit rechercher et poursuivre une cible mobile. Ce comportement s'identifie à celui d'un prédateur en présence d'une proie. Les seules informations que nous détenons sont les observations faites par le robot et sa distance relative par rapport au but.
- Le dernier chapitre est consacré à la conclusion de notre rapport ainsi qu'à la présentation des perspectives que nous aimerions développer par la suite.

2. Planification et modèles stochastiques

2.1. Introduction

Cette partie est consacrée à la présentation du centre d'intérêt de ce rapport qui est l'utilisation des modèles stochastiques pour la robotique mobile et des différents modèles utilisés en s'appuyant sur les définitions données dans [Laroche, 2000]. Dans un premier temps, nous exposerons les modèles stochastiques auxquels nous nous sommes particulièrement intéressés. Après avoir introduit quelques définitions, nous aborderons les Processus Décisionnels de Markov Parfaitement Observés, cas particuliers des Processus Décisionnels de Markov Partiellement Observés.

2.2. Les processus stochastiques

Processus stochastique

Un processus stochastique est une famille de variables aléatoires $X(t)$ où t est un paramètre réel prenant ses valeurs dans ensemble T. En général, T est dénombrable et représente le temps, le processus est alors dit discret.

Suite stochastique

Soit un système pouvant se trouver dans un ensemble dénombrable d'états S = $\{s_1, s_2,\ldots, s_N\}$. Entre les instants t et $t + 1$, le système passe aléatoirement de l'état s_i à l'état s_j. Pour représenter ce processus aléatoire, nous définissons la variable q_t de la façon suivante : $q_t = s_i$ signifie que le système est dans l'état s_i au temps t. Par définition l'ensemble $(q_1, q_2,\ldots, q_t,\ldots)$ est une suite stochastique à ensemble discret d'états.

Chaîne de Markov

Une suite stochastique vérifie la propriété de Markov si pour tout t :

$$Prob(q_t = s_i \mid q_{t-1} = s_j, q_{t-2} = s_k, q_{t-3} = \ldots) = Prob(qt = s_i \mid q_{t-1} = s_j)$$

Cette propriété peut être explicitée de la façon suivante : l'état du système à l'instant t dépend uniquement de l'état à l'instant $t - 1$. Nous pouvons la généraliser aux suites stochastiques d'ordre n, pour lesquelles l'état à l'instant t dépend des n précédents états.

Enfin, une chaîne de Markov est dite stationnaire si la probabilité de transition entre états est indépendante du temps, plus formellement si pour tout t et k :

$$Prob(q_t = s_i \mid q_{t-1} = s_j) = Prob(q_{t+k} = s_i \mid q_{t+k-1} = s_j)$$

2.3. Les Processus Décisionnels de Markov

2.3.1. Cadre théorique

Les incertitudes vont être gérées en utilisant le Processus Décisionnel de Markov Parfaitement Observé (que nous désignerons dans la suite par le terme de PDM) [Bellman,1957]. Il peut être défini comme une formalisation mathématique de problèmes dans lesquels le robot doit décider comment agir afin de maximiser une fonction de gain. En utilisant le PDM, le robot connaît à chaque instant sa position exacte au sein de l'environnement.

Ainsi un PDM est défini par un tuple « S,A,T,R ».

- S : ensemble fini d'états de l'environnement parfaitement identifiables ;
- A : ensemble fini d'actions ;
- T : fonction de transition entre états selon l'action effectuée : $T : S \times A \times S \rightarrow [0,1]$

Nous notons $T(s,a,s')$ la probabilité de passer de l'état s à l'état s' en effectuant l'action a.

- R : fonction de gain qui permet de déterminer le (les) but(s) à atteindre et les éventuelles zones dangereuses de l'environnement. Cette fonction peut être définie de différentes manières, suivant le problème à résoudre. Dans la suite de cette partie, nous utiliserons le formalisme : $R : S \rightarrow \Re$.

L'hypothèse de complète observabilité de l'état courant peut à priori sembler très restrictive, mais elle sera supprimée lors de l'exécution d'un plan. Cette simplification a le grand avantage de permettre le calcul de plans sur un espace d'états fini. Ainsi, nous réduisons considérablement la complexité des algorithmes. Pour ce type de modèle, nous parlerons aussi de politique pour définir un plan.

2.3.2. Politiques pour des Processus Décisionnels de Markov

Critère d'optimalité

La politique optimale est calculée suivant la fonction de gain : le but est d'optimiser les récompenses possibles. Nous distinguons deux critères : la récompense moyenne (*average-reward criterion*) et la récompense totale pondérée (*discounted infinite horizon reward*). Ce dernier critère est le plus utilisé et il est très intéressant. Puisqu'il permet de faire un compromis entre la récompense à court terme (conséquence immédiate de l'exécution d'une action) et la récompense qui pourra résulter de l'exécution de cette action mais à plus long terme.

Dans ce cas l'espérance de gain est formulée ainsi :

$$E\left(\sum_{t=0}^{\infty} \gamma^t \cdot r_t\right) \qquad (2.1)$$

Ici r_t représente la récompense obtenue à l'instant t. Le coefficient γ, introduit dans le critère d'optimalité, permet de faire le compromis voulu. Ce paramètre est un réel compris entre 0 et 1. Il assure la convergence de l'algorithme. Aussi, si nous faisons varier ce paramètre, nous obtenons différents types de politiques, chacune ayant son propre comportement. Donc dans la suite de ce rapport, nous utiliserons le critère d'optimalité de récompense totale pondérée.

Forme générale

Puisque l'espace d'états est fini et que chaque état est parfaitement identifiable par rapport aux autres, une politique π affecte une action à un état ($\pi : S \rightarrow A$). Calculée sur un horizon infini, la politique optimale est stationnaire, c'est-à-dire qu'elle est indépendante du temps.

2.3.3. Algorithmes de calcul de politiques

Dans le cadre des PDM, il existe plusieurs algorithmes permettant de calculer une politique optimale. Lors de ce stage nous avons utilisé les deux plus connus : *Value Iteration*, décrit dans [Bellman, 1957] et *Policy Iteration* introduit trois ans plus tard par [Howard, 1960].

L'algorithme Value Iteration

Cet algorithme apparemment complexe est en fait très simple, il permet de calculer itérativement la valeur de chaque état. La valeur d'un état est en fait le gain obtenu par l'exécution d'une action auquel nous ajoutons les valeurs des différents états qu'il est possible d'atteindre en exécutant cette même action. De plus, nous prenons en compte le facteur γ permettant d'introduire un compromis entre action à court terme et action à long terme. Plus formellement, nous pouvons définir la valeur d'un état à l'instant t ainsi :

$$V_t(s) = max_a \left[R(s) + \gamma \sum_{s' \in S} T(s,a,s') \text{ x } V_{t-1}(s')\right] \qquad (2.2)$$

Comme il permet un calcul itératif, nous pouvons le classer parmi les algorithmes *anytime*[1].

L'algorithme montre plus précisément comment cette valeur est calculée récursivement sur tous les états.

[1] Un algorithme anytime est un algorithme itératif qui garantit de produire une réponse à toute étape du calcul, où la réponse est supposée s'améliorer à chaque itération.

Algorithme de *Value Iteration* :

// Initialisation des états à une valeur nulle
$t \leftarrow 0$
pour tout $s \in S$ **faire**
    $V_0(s) = 0$
fin pour

// Calcul récursif des valeurs jusqu'au passage sous un seuil
répéter
    $t \leftarrow t + 1$
    pour tout $s \in S$ **faire**
        $V_t(s) = max_a [R(s) + \gamma \Sigma_{s' \in S} T(s,a,s') \times V_{t-1}(s')]$
        $\pi_t(s) = argmax_a [R(s) + \gamma \Sigma_{s' \in S} T(s,a,s') \times V_{t-1}(s')]$
    fin pour
jusqu'à $max_s |V_t(s) - V_{t-1}(s)| < \varepsilon$
retourner π_t

Nous initialisons tout d'abord une valeur nulle et une récompense constante à chaque état en fonction de sa nature (but, obstacle, état libre). Puis, nous calculons successivement les valeurs de chaque état à l'instant t en utilisant les valeurs au temps t-1. Nous savons qu'une politique est optimale à l'horizon t quand le processus s'arrête après l'exécution de t actions. Par conséquent pour obtenir une politique à l'horizon infinie, nous itérons jusqu'à ce que les valeurs des états ne varient plus que de façon minime en utilisant un seuil ε. La politique obtenue est alors sous-optimale mais plus ε est proche de 0, plus les valeurs des états sont proches des valeurs optimales. A chaque itération, nous déterminons pour chaque état l'action permettant d'obtenir la valeur la plus intéressante afin d'améliorer la politique.

Complexité de l'algorithme

L'efficacité de l'algorithme dépend de deux facteurs : la complexité d'une itération, et le nombre d'itérations nécessaire pour converger. Chaque itération consiste à calculer la valeur de transition entre les états pour chaque action : cela nécessite $|A||S|^2$ opérations. Le nombre d'itérations nécessaire est plus difficile à déterminer. [Littman et al., 1995b] montrent que l'algorithme est polynomial selon $|S|$, $|A|$, γ et B, où B est le nombre de bits nécessaires à la représentation des données du problème. Nous donnerons un exemple d'exécution de cet algorithme dans le paragraphe 3.3**.**

L'algorithme Policy Iteration

Ce deuxième algorithme est plus complexe. Il s'agit également d'un algorithme itératif applicable à chaque pas de calcul jusqu'à obtenir la politique optimale. Contrairement à *Value Iteration* qui calcule une politique sans connaissance à priori, cet algorithme nécessite l'initialisation d'une première politique quelconque, que *Policy Iteration* va ensuite améliorer par itérations successives.

Chaque itération se décompose en deux phases. Tout d'abord une première phase d'évaluation de la politique courante π à l'aide de la formule suivante :

$$V_\pi(s) = R(s) + \gamma \sum_{s' \in S} T(s, \pi(s), s') \times V_\pi(s') \qquad (2.3)$$

Cette phase revient donc à la résolution d'un système de $|S|$ équations à $|S|$ inconnues. Enfin la seconde phase consiste à améliorer cette politique courante en déterminant pour chaque état une action améliorant la valeur courante, un peu comme dans *Value Iteration*. La politique optimale est obtenue lorsque deux politiques successives sont identiques et nécessairement optimales.

Algorithme de *Policy Iteration* :

// Initialisation avec une politique quelconque π_i
$\pi' \leftarrow \pi_i$

// Calcul itératif jusqu'à obtenir deux politiques identiques
répéter
  $\pi \leftarrow \pi'$
  // Phase d'évaluation de la politique courante
  pour tout $s \in S$ **faire**
    Calculer $V\pi(s)$ en résolvant les $|S|$ équations à $|S|$ inconnues suivant l'équation 2.3
  fin pour

  // Phase d'amélioration
  pour tout $s \in S$ **faire**
    si il existe une action $a \in A$ telle que :
      $R(s) + \gamma \sum_{s' \in S} T(s,a,s') \times V_\pi(s') > V_\pi(s)$
    alors $\pi'(s) \leftarrow a$
    sinon $\pi'(s) \leftarrow \pi(s)$
    finsi
  fin pour
jusqu'à $\pi = \pi'$
retourner π

Complexité de cet algorithme.

Chaque itération de l'algorithme consiste en deux opérations : la résolution du système d'équations qui nécessite un peu moins de $|S|^3$ opérations et la phase d'amélioration qui est effectuée en $|A|\ |S|^2$ opérations. Comme précédemment, le nombre d'itérations nécessaire à la convergence de l'algorithme est plus difficile à déterminer. [Littman et al., 1995b] donnent le même résultat que pour *Value Iteration*.

Comparaison des deux algorithmes

Les deux algorithmes ont un fonctionnement très différent :

- *Value Iteration* procède par de petites améliorations successives de la politique. En pratique, cet algorithme nécessite un grand nombre d'itérations pour converger mais chaque itération est très rapide.
- *Policy Iteration* améliore beaucoup la politique à chaque itération. Généralement, le nombre d'itérations nécessaire à la convergence est faible mais chaque itération est très coûteuse.

Afin de combiner les avantages des deux algorithmes, certains travaux ont essayé de créer un algorithme hybride présenté sous le nom de Modified Policy Iteration [Puterman, 1994]. Mais il est encore difficile aujourd'hui de déterminer quel est l'algorithme le plus rapide. [Littman, 1996] donne différentes références, chacune affirmant la supériorité d'un algorithme par rapport à un autre. La conclusion semble être que le choix de l'algorithme doit être fait en fonction de l'application après évaluation. C'est ainsi que nous avons procédé, comme nous le verrons au paragraphe 3.3.

Autres approches de résolution

Vu la complexité des algorithmes présentés, de nombreux travaux se sont penchés sur l'utilisation de méthodes permettant d'accélérer ces calculs. Mais une approche peut néanmoins être brièvement décrite ici, l'utilisation d'apprentissage par renforcement.

Ce type d'algorithmes peut être utilisé non seulement pour apprendre la politique optimale, mais aussi pour apprendre la fonction de transition si celle-ci n'est pas connue. Parmi les algorithmes d'apprentissage par renforcement, nous pouvons citer le Q-learning [Watkins et Dayan, 1992] qui est le plus connu. Malheureusement ces algorithmes sont souvent très complexes et l'obtention de la politique optimale peut être très lente : la preuve de convergence de l'algorithme nécessite que chaque paire (état, action) ait été testée une infinité de fois.

2.4. Conclusion

Les modèles présentés ont le grand intérêt de permettre la prise en compte des diverses incertitudes dégradant la localisation du robot. En utilisant ces modèles, nous pouvons espérer obtenir des plans optimaux prenant en compte le risque d'échec et essayant de le minimiser. Ainsi, nous trouverons un compromis, particulièrement intéressant en robotique, entre l'efficacité et la sécurité.

Aussi, ce cadre théorique met en œuvre suffisamment de paramètres pour que nous puissions l'adapter à notre application et développer un prototype réellement intéressant.

Le chapitre suivant montre comment nous avons défini ces différents paramètres, pour que notre PDM soit en accord avec nos objectifs.

3. Processus Décisionnels de Markov et robotique

3.1. Introduction

Ce n'est qu'au début des années 90, que la communauté « intelligence artificielle » a commencé à s'intéresser aux Processus Décisionnels de Markov. Depuis, cet intérêt n'a cessé, comme le montre l'article de synthèse paru dans la revue *Artificial Intelligence* faisant le point de ce qui a été fait sur la planification décisionnelle et théorique [Boutilier *et al.*, 1999] et la toute récente thèse de Pierre Laroche sur les *PDM appliqués à la planification sous incertitude* soutenue en janvier 2000 [Laroche, 2000].

Un Processus Décisionnel de Markov est un modèle qui peut être paramétré de diverses manières et nous devons effectuer des choix afin de passer du modèle théorique à son application à la robotique mobile. Ainsi, la façon de représenter l'environnement du robot (les états), les buts ou les actions possibles sont des problèmes classiques. De plus, un PDM doit être paramétré au niveau de ces éléments fondamentaux comme les fonctions de gain et de transition, le traitement des obstacles ou encore le critère d'optimalité et la valeur de γ.

3.2. Un Processus Décisionnel de Markov adapté à la robotique

3.2.1. Définitions des états et des actions utilisées

En robotique, la mission à remplir est généralement de se déplacer efficacement et rapidement dans un environnement pour atteindre une position précise. De ce fait, les états définissent généralement des positions géographiques. Ainsi, la décomposition de l'environnement par une grille de cases de même taille semble idéale comme représentation. Dans cette représentation chaque case correspond à un état.

La figure 3.1 présente un environnement possible où une case définit donc un état. La couleur du cercle détermine la nature de l'état, à savoir en noir pour un obstacle, en jaune pour un but et en couleur de fond s'il s'agit d'un état libre.

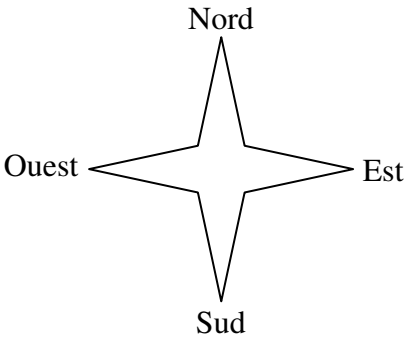


Figure 3.0 *– Boussole*

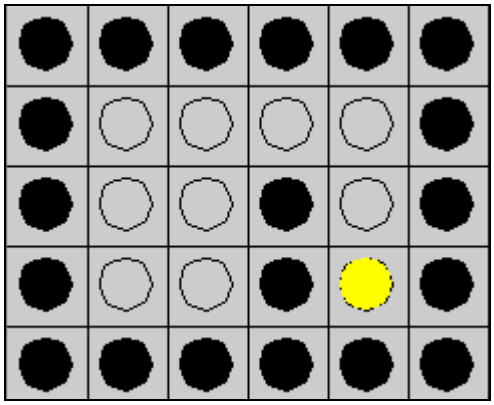


Figure 3.1 *– Petit environnement (9 états libres)*

Comme pour les états, afin de conserver un niveau de précision nécessaire à l'exécution de plans, nous contrôlons le robot à travers quatre actions atomiques : Avancer Nord d'une case, Avancer Sud d'une case, Avancer Est d'une case et Avancer Ouest d'une case. Ces actions sont simples et peuvent être facilement programmées comme commandes de base pour un robot.

3.2.2. Fonction de gain et traitement des obstacles

Comme nous l'avons déjà vu au chapitre précédent, cette fonction permet de désigner le (ou les) but(s) à atteindre et de spécifier les zones dangereuses de l'environnement. Les états constituant une zone dangereuse se voient allouer une punition et les états contenant le (ou les) but(s) à atteindre une récompense. Mais que faire pour les états n'entrant pas dans ces deux catégories ? A priori, plus l'état est proche du but, plus la récompense doit être forte. Cette hypothèse nous donne une fonction de gain qui peut être calculée en fonction de la distance entre l'état considéré et le but. Mais si nous examinons la façon dont est calculée la politique optimale permettant d'atteindre le but, nous nous apercevons que la fonction de gain peut être beaucoup plus simple.

Si la fonction de gain affecte une récompense forte au fait de se trouver dans l'état but, alors les états à partir desquels le robot peut accéder directement au but auront une valeur forte. Les états adjacents à ces états auront une valeur un peu moins forte, et ainsi de suite, la valeur diminuant au fur et à mesure que nous nous éloignons du but. Ainsi, il apparaît, nécessaire et suffisant, de distinguer le gain obtenu au but des gains obtenus dans les autres états pour obtenir une politique optimale. Ceci permet de générer une fonction de gain très simple, utilisée d'ailleurs dans [Dean et al.,1993] :

$$R(s) = \begin{cases} 0 \text{ si s est l'état but} \\ -1 \text{ sinon} \end{cases}$$

Remarque 1 : afin de s'assurer que le but ait la valeur la plus forte possible (c'est-à-dire 0), il est défini comme un état absorbant : une fois le but atteint, il est impossible d'en sortir, quelle que soit l'action exécutée.

Remarque 2 : si la fonction de transition est déterministe, une fonction de gain de ce type permet d'obtenir des politiques optimales selon le critère du plus court chemin.

Cette fonction très simple est satisfaisante.

Lors de l'exécution d'une suite d'actions dans un environnement réaliste, un robot mobile peut rencontrer plusieurs types d'obstacles. Ils peuvent être mobiles ou non, connus à l'avance ou non. Si l'évitement d'obstacles inconnus ou mobiles ne peut être considéré que lors de l'exécution d'un plan, il est évident que les politiques doivent prévoir d'éviter ceux connus. L'utilisation d'une valeur négative pour les obstacles, les rendant ainsi repoussants, permet également de ne pas ajouter à la complexité du problème le calcul de leurs valeurs par les algorithmes. Cette solution permet de donner une valeur faible aux obstacles, tout en conservant la simplicité de la fonction de gain. La valeur d'un état repoussant (donc d'un obstacle) *s* est la suivante :

$$V_\pi(s) = R(s) + \gamma(1 \times V_\pi(s)) \Rightarrow V_\pi(s) = \frac{-1}{1-\gamma}$$

La simplicité de notre fonction de gain et du traitement des obstacles a encore facilité la programmation. En affectant une valeur identique pour chaque obstacle et chaque but, elles bornent strictement les valeurs des états libres. Ainsi, dans les algorithmes détaillés dans la partie précédente, les calculs se font uniquement sur les états libres, facilement identifiables par leurs récompenses fixées à –1.

3.2.3. Fonction de transition

La fonction de transition est un élément fondamental dans l'application des Processus Décisionnels de Markov. En effet, cette fonction prend en compte l'incertitude quant aux conséquences des actions. Dans la plupart des travaux alliant MDP et robotique [Cassandra et al., 1996] [Dean et al., 1993], cette fonction est considérée comme une donnée du problème, ou bien elle est générée expérimentalement.

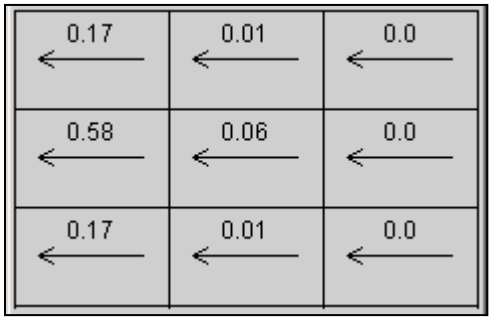


Figure 3.2 *– Fonction de transition pour l'action avancer ouest*

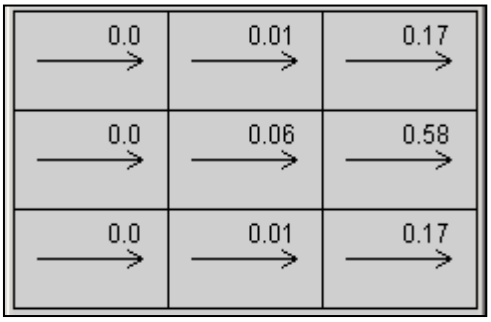


Figure 3.3 *– Fonction de transition pour l'action avancer est*

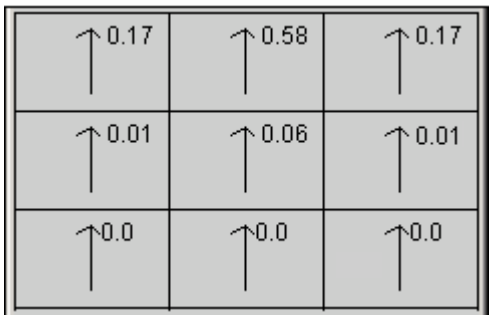


Figure 3.4 *– Fonction de transition pour l'action avancer nord*

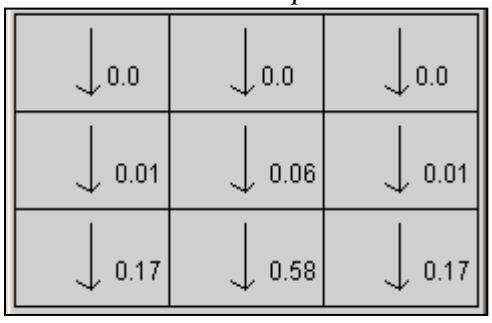


Figure 3.5 *– Fonction de transition pour l'action avancer sud*

Pour les fonctions de transition que nous avons illustrées dans les figures 3.2, 3.3, 3.4 et 3.5, nous avons choisi neuf cases. Elles représentent les cases adjacentes à la case du robot (sachant que la case du robot est la case qui se trouve au centre), dans lesquelles le robot peut éventuellement se trouver après avoir effectué une action parmi les quatre actions possibles. Pour mieux comprendre la fonction de transition, nous allons détailler

celle définit pour l'action avancer Ouest. Tout en sachant que les fonctions de transitions pour les trois autres actions se basent sur le même principe, la seule différence étant le changement d'orientation.

Nous allons numéroter les neuf cases comme illustrer dans la figure 3.6 et cette représentation reste identique pour toutes les fonctions de transition.

1	2	3
4	5	6
7	8	9

***Figure 3.6** – Représentation des neuf cases*

Maintenant attardons-nous sur la figure 3.2. Avant d'exécuter l'action, le robot se trouve dans la case centrale ou d'après la figure 3.6 dans la case 5. La probabilité que le robot reste dans la case 5, sachant que son action a été d'avancer vers l'ouest et que sa direction initiale est quelconque, est égale à 0.06. Prenons encore un exemple. La probabilité que le robot se trouve dans la case 4, en effectuant l'action avancer vers l'ouest est égale à 0.58. Le raisonnement est le même pour les autres cases.

Enfin, dans notre volonté de développer une application modulable et flexible, nous avons intégré la possibilité de modifier, pour chaque action, les valeurs des probabilités de notre fonction de transition mais uniquement pour les effets des actions déjà possibles.

3.2.4. Critères d'optimalité et valeur de γ

Nous utilisons la fonction de gain définie précédemment pour calculer une politique qui permettra d'atteindre un but fixé. Mais à une fonction de gain peut correspondre diverses stratégies optimales en fonction du critère d'optimalité (voir le paragraphe 2.3.2). En effet, si nous nous intéressons aux récompenses immédiates, il suffit de choisir en chaque état, l'action qui mène vers l'état le plus intéressant (c'est-à-dire le plus récompensé). Ce genre de comportement est évidemment peu efficace, il faut prendre en compte les conséquences des actions à plus long terme. Parmi les critères d'optimalité, le plus souvent utilisé est celui qui cherche à maximiser l'espérance pondérée de gain futur (*discounted infinite horizon reward*). Il donne plus de portée aux effets immédiats et plus les conséquences sont lointaines, moins elles sont importantes.

Ce critère est assez naturel en planification. Quand une personne, se rendant à son domicile distant de dix kilomètres, fait un détour de 100 mètres ce n'est pas bien grave. En revanche, si elle se trouve sur le trottoir d'en face et que le plan lui fait faire un détour de 100 mètres, c'est beaucoup plus fâcheux. Dans le premier cas la personne est loin du but, l'effet du détour est peu important mais dans le second cas il a un effet néfaste immédiat.

Ce critère d'optimalité permet de trouver un compromis entre les effets immédiats et les effets plus lointains, donc nous allons l'utiliser par la suite.

Le paramètre γ permet d'accorder plus ou moins d'importance aux gains futurs. Pour un même environnement, la valeur de ce paramètre influe sur les différents plans obtenus. Nous pouvons illustrer cette remarque à l'aide du petit environnement présenté auparavant (figure 3.1). La figure 3.7 montre deux politiques calculées pour cet environnement en utilisant deux valeurs de γ différentes : 0.7 à gauche et 0.95 à droite. L'action avancer est représentée par une flèche reliant un cercle à un autre. Elle est orientée vers le nord si nous effectuons l'action avancer Nord, vers le sud si l'action est avancer sud, vers l'ouest si l'action est avancer Ouest et vers l'est si l'action est avancer est.

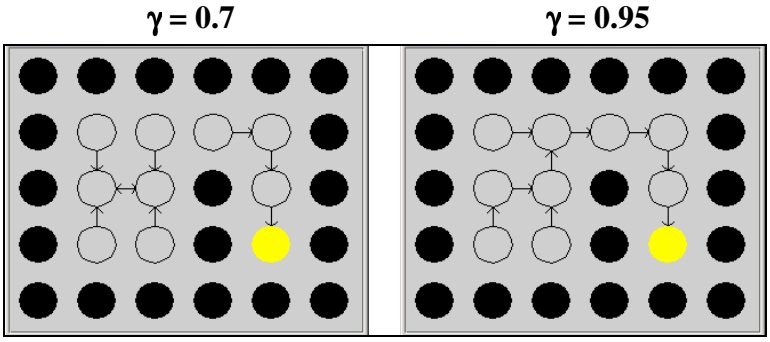


***Figure 3.7** – Politiques obtenues pour différentes valeurs de γ*

Ces deux politiques diffèrent pour les trois positions en haut : (1,1), (1,2) et (2,2). En effet sur la figure de gauche et dans l'état (2,2), nous pouvons voir que le robot préfère avancer à l'ouest plutôt que d'avancer au nord.

Les politiques doivent à la fois être sûres et efficaces. En conséquence, les chemins choisis doivent présenter un compromis selon la distance à parcourir et la sécurité du robot sur ces chemins. Un chemin court mais obstrué, dans lequel le robot devra avancer lentement pour se faufiler entre les obstacles, est dans certains cas moins intéressant qu'un chemin plus long dénué d'obstacles, dans lequel le robot pourra se déplacer rapidement et à moindre risque.

En fait, selon les environnements et leurs tailles, une politique satisfaisante - c'est à dire sans collisions et joignant le but à partir de toutes positions - ou optimale est obtenue avec une valeur de γ assez proche de 1. Ainsi dans tous les exemples qui suivent nous avons fixé la valeur de γ à 0.99, car elle permet d'obtenir des plans satisfaisants dans le type d'environnement que nous utilisons. Il faut également noter que plus γ est proche de 1 plus le nombre d'itérations nécessaires à l'obtention du plan est grand.

Enfin, notre prototype étant une application, cette valeur doit être facilement modulable et ceci pour chaque environnement afin de pouvoir comparer les différentes politiques obtenues.

3.3. La planification

Les deux algorithmes classiques de résolution de Processus Décisionnel de Markov ont chacun leurs supporters : selon Puterman [Puterman, 1994] , Value Iteration est plus rapide, mais Tijms [Tijms, 1986] favorise Policy Iteration. En fait, les deux algorithmes ont des fonctionnements très différents : Value Iteration nécessite de nombreuses mais rapides itérations, alors que Policy Iteration converge au bout de très peu d'itérations, mais chacune d'entre elles nécessite beaucoup de temps. Le choix de l'algorithme pourrait dépendre de la structure du MDP à résoudre, mais il n'existe pas de méthode permettant de choisir le meilleur algorithme en fonction du type de MDP. Afin d'illustrer le fonctionnement de ces algorithmes et de motiver notre choix, nous comparons les deux algorithmes sur un petit exemple à 9 états (figure 3.1) dans les paragraphes suivants.

Policy Iteration : politiques successives

Le principe de Policy Iteration, vu au paragraphe 2.3.3, est très simple : partant d'une politique initiale quelconque, l'algorithme cherche à maximiser la valeur de chaque état et itère jusqu'à obtenir deux politiques successives identiques, qui sont alors optimales. La figure 3.8 montre les politiques obtenues à chaque itération d'une exécution de l'algorithme. La politique initiale est totalement aléatoire, sans chercher à éviter les obstacles. Une fois que nous connaissons la valeur de chaque état de la politique courante, nous regardons si la valeur d'états pourrait être améliorée en changeant l'action effectuée. Dès la première itération, la politique est fortement améliorée. Chaque politique ne peut être qu'une amélioration par rapport à la précédente (à moins que celle-ci ne soit déjà optimale). La politique optimale est obtenue après la deuxième itération. La troisième itération est nécessaire puisque la condition d'arrêt de l'algorithme est l'obtention de deux politiques successives identiques.

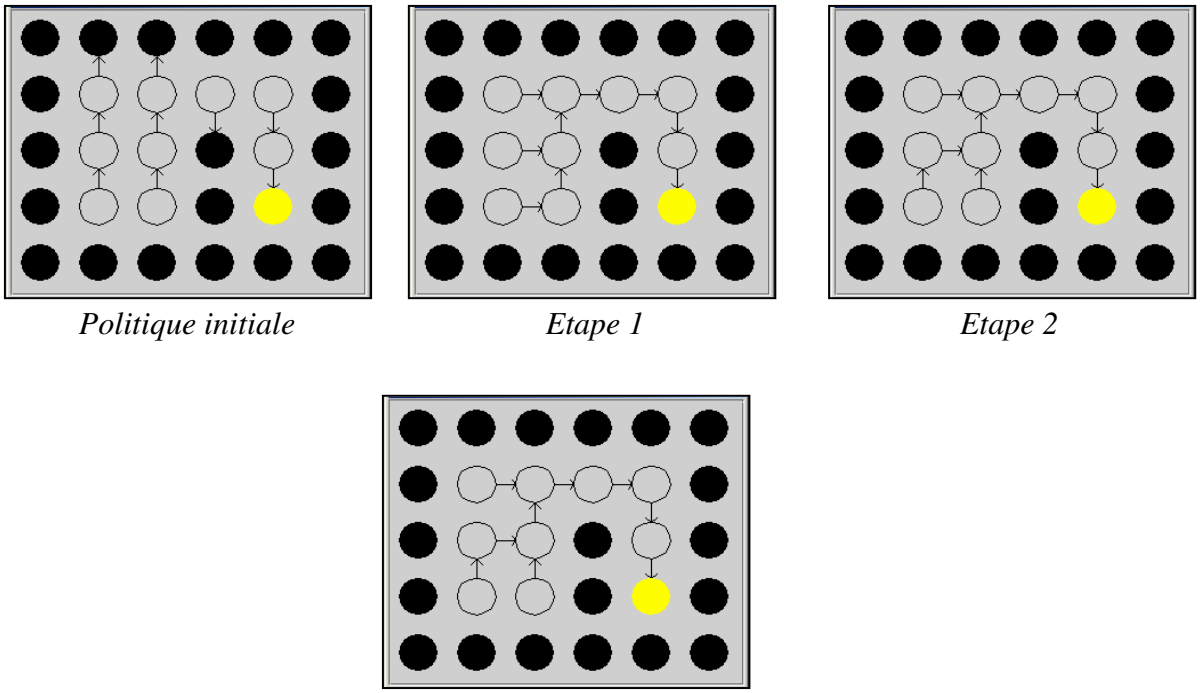


Figure 3.8 *– Policy Iteration : d'une politique aléatoire à la politique optimale*

Cette figure ne montre qu'un exemple de déroulement de l'algorithme. En effet, il dépend de la politique initiale qui peut être plus ou moins bonne.

Value Iteration : politiques successives

L'algorithme Value Iteration a un fonctionnement très différent, plus « exploratrice » (voir le paragraphe 2.3.3). En effet, nous cherchons à approcher l'espérance de gain de chaque état sur un horizon infini. Au départ, tous les états ont une valeur nulle. Lors de la première itération, nous calculons la valeur obtenue par l'exécution d'une seule action par état. La deuxième itération donne la valeur à l'horizon 2, c'est-à-dire si nous exécutons successivement deux actions, et ainsi de suite. La condition d'arrêt de l'algorithme est donc définie par un seuil de différence entre deux valeurs successives. Dans l'exemple donné ici, l'algorithme s'arrête quand la différence de valeur totale de deux politiques successives est inférieure à 10^4.

La figure 3.9 montre les trois premières itérations de l'algorithme.

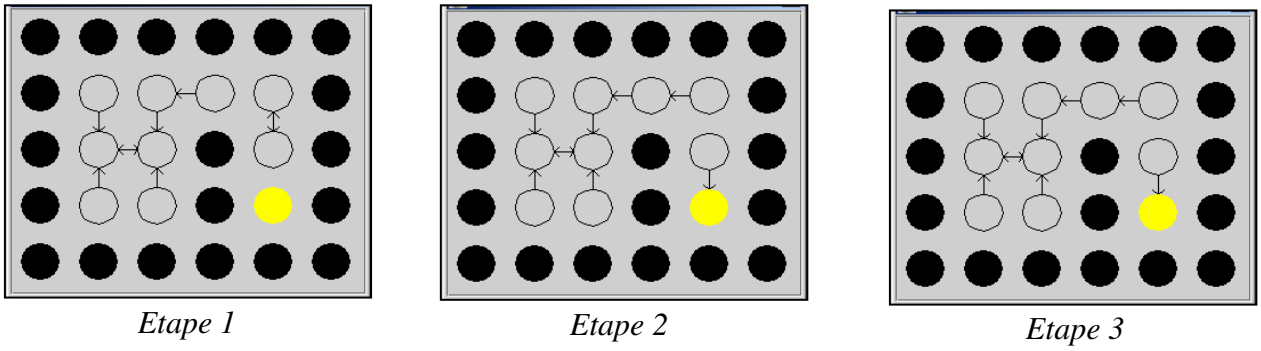
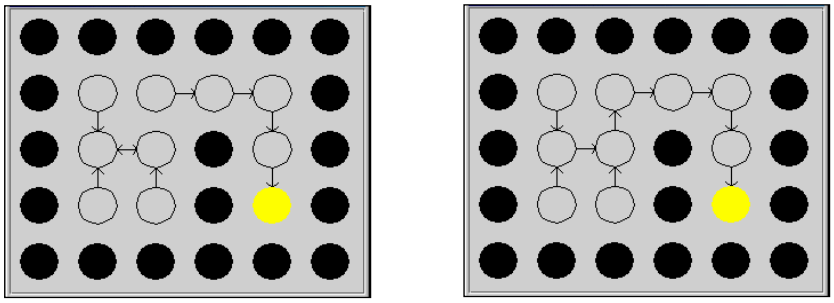


Figure 3.9 *– Value Iteration : les trois itérations initiales.*

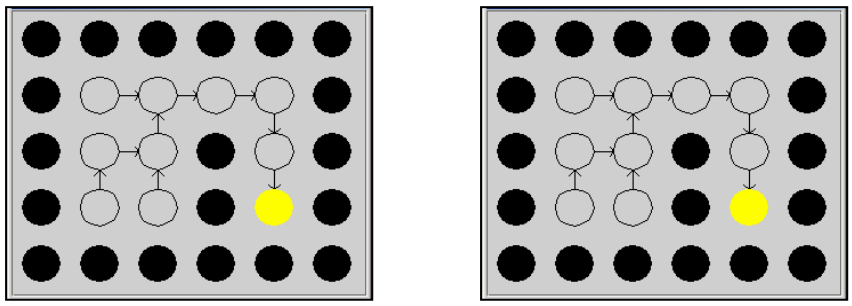
Nous pouvons ainsi remarquer dans la figure 3.9 que les trois politiques sont quasiment identiques. La figure 3.10 montre les premiers progrès de la politique qui n'interviennent qu'à l'itération 15 de l'algorithme. A chaque itération un choix est effectué entre les quatre actions de translation qui permet au robot de se rapprocher du but tout en essayant au maximum d'éviter les collisions.



Etape 15 *Etape 16*

***Figure 3.10** – Value Iteration : Améliorations.*

Au fur et à mesure que le nombre d'itérations augmente, la politique est améliorée. C'est enfin après la 23ème itération que la politique optimale est trouvée (voir figure 3.11).



Etape 23 *Etape 24*

***Figure 3.11** – Value Iteration : obtention de la solution optimale*

Ainsi, dans l'exemple illustré précédemment, nous avons pu montrer que l'algorithme Value Iteration, procède par de petites améliorations successives de la politique.

Ce petit exemple ne constitue en aucun cas une preuve de la supériorité de Policy Iteration vis-à-vis de Value Iteration. Sur ce même problème, en faisant uniquement passer le facteur γ de 0.99 à 0.9, l'écart entre les deux algorithmes a fortement diminué. Le nombre d'itérations de Value Iteration a baissé et s'effectue en 1,7 secondes, alors que le temps de calcul de Policy Iteration n'a pas varié. Ces constatations montrent l'importance des paramètres du PDM sur l'efficacité de ces algorithmes.

Dans sa thèse [Laroche, 2000], Pierre Laroche montre l'intérêt d'initialiser l'algorithme *Policy Iteration* avec une politique plus « intelligente » permettant de converger plus rapidement vers le plan optimal. En effet, d'après ces résultats, l'initialisation de l'algorithme par une politique du plus court chemin plutôt que par une politique aléatoire - souvent « mauvaise » - a permis de réduire le temps de calcul d'environ 25%.

Malgré l'hypothèse de départ quant à la parfaite observabilité de l'état courant, peu réaliste dans le cadre de la robotique mobile, il existe des approches utilisant les Processus Décisionnels de Markov dans ce cadre, même s'il ne s'agit pas tout à fait de « vrais » Processus Décisionnels de Markov. En effet, si la politique est calculée à l'aide des algorithmes précédents, leur exécution est quant à elle supervisée par des techniques plus proches des Processus Décisionnels de Markov Partiellement Observés (PDMPO). Cette technique, après l'exécution d'une action et après une observation[2], calcule la distribution de probabilités sur les états. A partir de cette distribution, il existe de nombreuses stratégies permettant de déterminer l'action à exécuter [Cassandra et al.,1996]. Nous allons détailler cette technique un peu plus loin.

3.4. L'exécution

Une politique donnée par la résolution d'un PDM est un ensemble de couples (état, action) spécifiant dans chaque état une action optimale à exécuter. Cependant, lors de l'exécution d'une politique, la position courante du robot est très rarement connue avec certitude à cause des incertitudes liées aux actions.

Pour pallier ce problème, un robot est muni de capteurs pour percevoir son environnement et s'y repérer[3]. Malheureusement, ces capteurs renvoient des données bruitées qui rendent la localisation très incertaine. Plus particulièrement, dans chaque environnement nous aurons toujours des positions différentes pour lesquelles les observations du robot seront identiques. Par exemple, si nous considérons un couloir de laboratoire comme environnement du robot, il ne pourra distinguer dans quelle partie de couloir, face à quelle porte ou encore dans quel bureau il se trouve.

C'est pourquoi, une bonne exécution des politiques ne peut être envisagée sans l'ajout de différents modules gérant la localisation du robot.

Pour résoudre ce problème, il existe une technique appelée localisation markovienne [Dieter et al.,1999]. Elle calcule au fur et à mesure de l'exécution du plan, une distribution de probabilités sur les états de l'environnement. Cette technique est en fait celle utilisée dans les PDMPO[4] pour l'exécution d'un plan. Pour cela, elle nécessite un modèle d'incertitude liée aux actions, présenté précédemment, mais aussi un modèle d'incertitude liée aux données bruitées des capteurs (appelé généralement fonction d'observation ou modèle capteur). Pour finir, il faut définir une stratégie permettant de choisir l'action à effectuer en fonction de cette distribution.

Tout d'abord, nous définissons formellement la fonction d'observation. Puis, nous expliquons le fonctionnement de la localisation markovienne et nous présentons diverses stratégies de choix de l'action en fonction de la distribution de probabilités sur les états.

---

[2] Une observation est une donnée symbolique obtenue par l'interprétation des données brutes des capteurs.

[3] Cette opération est généralement appelée la localisation.

[4] Rappelons que dans un PDMPO, les algorithmes de planification associent une action, non pas à un état, mais à une distribution de probabilités sur les états. Donc, dans ce type d'approche, il faut connaître à chaque instant la distribution des probabilités sur les états pour savoir l'action à exécuter. De plus, contrairement aux algorithmes de planification associés aux PDMPO, PDM utilise des algorithmes d'une complexité polynomiale, qui les rendent tout à fait utilisables dans la pratique.

3.4.1. Fonction d'observation

Soit ϑ l'ensemble fini de symboles observables. La fonction d'observation O, spécifie la probabilité d'observer un symbole de ϑ connaissant l'état dans lequel se trouve le système, dans notre cas le robot. Ainsi la fonction d'observation est définie par : O : S x $\vartheta \rightarrow$ [0,1], où O(s,o) correspond à la probabilité d'observer o dans l'état s.

3.4.2. Algorithme de localisation

Pour résoudre ce problème, nous utilisons généralement un ensemble d'états probables (nous parlons alors de *belief state*) qui contient les états dans lesquels le robot peut éventuellement se trouver. C'est ainsi que l'ensemble des états libres et des buts de l'environnement constituent les états probables de localisation du robot, ensemble encore appelé distribution. Nous nous situons donc à la frontière des POMDP, puisque cet ensemble est le même que celui utilisé pour représenter l'espace d'états dans ce cadre.

Nous pouvons avoir deux cas possibles : le premier si la position initiale du robot est connue et l'autre si la position initiale du robot est inconnue. A partir de ces deux options, nous pouvons facilement déterminer la distribution de probabilités des états à $t = 0$. Ainsi, si la position initiale du robot dans l'état s_0 est connue, alors $Pr_{t=0}(s_0) = 1$ et pour tout s différent de s_0, $Pr_{t=0}(s) = 0$. Cependant, si la position initiale du robot est inconnue, les probabilités de chaque état s de l'environnement sont équiprobables : pour tout s $Pr_{t=0}(s) = 1/|S|$.

Par conséquent, la connaissance de la position initiale du robot est déterminante et les deux possibilités donneront des exécutions souvent différentes pour un même point de départ. C'est pourquoi nous avons directement distingué ces deux cas dans le menu des fonctionnalités principales de notre application.

L'algorithme de localisation fonctionne de la même manière pour la position initiale connue ou inconnue. Ainsi au pas initial, il faut prendre les observations du robot. Puis nous calculons la distribution, nous exécutons l'action correspondante et nous reprenons les observations faites par le robot. Nous répétons cette phase jusqu'à ce que le robot arrive au but.

L'ensemble d'états probables est mis à jour en fonction des positions probables précédentes (la distribution au temps t-1), des données fournies par les capteurs (les observations du robot) et de la fonction de transition. La probabilité, notée $\Pr_t(s')$, de se trouver dans l'état s' au temps t alors que le robot a exécuté l'action a puis a observé l'observation o, est calculée de la manière suivante :

$$\Pr_t(s') = \frac{O(s',o) \times \sum_{s \in S} \Pr_{t-1}(s) \times T(s,a,s')}{\sum_{s \in S} \Pr_t(s)}$$

$Pr_{t-1}(s)$ est la probabilité que le robot soit dans l'état *s* au temps *t*-1.

$T(s,a,s')$ est la probabilité de passer de l'état *s* à l'état *s'* en effectuant l'action *a*.

$O(s',o)$ est la probabilité d'observer l'observation *o* (observation réelle du robot) dans l'état *s'*.

La distribution, est mise à jour après chaque action et chaque observation, ce qui permet d'ajuster la localisation du robot à chaque pas.

3.4.3. Choix pour l'exécution d'une action

Nous retrouvons la fonction O classiquement utilisée lorsque nous manipulons un MDP partiellement observé. Cet ensemble d'états va permettre de choisir l'action à exécuter. Pour cela, différentes stratégies sont possibles [Cassandra et al., 1996] : l'exécution de l'action correspondant à l'état ayant la plus forte probabilité, l'exécution de l'action la plus souvent optimale pour les états probables, etc. Dans les travaux étudiés, c'est quasiment toujours la première solution qui est choisie, nous avons donc fait de même.

3.5. Conclusion

Nous avons présenté dans cette partie l'adaptation des Processus Décisionnels de Markov à la robotique. Ensuite, nous avons étudié le modèle et les différents algorithmes. Enfin, nous avons présenté la localisation lors de la phase d'exécution.

Un des avantages des PDM que nous avons remarqué, est le fait qu'ils utilisent des algorithmes qui travaillent de façon itérative pour produire des résultats dont la qualité augmente avec le temps. La planification ne tient pas compte du fait que la position initiale du robot sera connue ou inconnue, elle est réalisée tout le temps de la même manière. Nous avons montré aussi quels facteurs sont importants pour obtenir des politiques à la fois efficaces et sûres. Les PDM nous offrent également la possibilité d'avoir plusieurs buts dans l'environnement. Nous avons pu correctement observer tous ces avantages à travers les exemples illustrés pendant ce chapitre.

Mais, les PDM présentent aussi des limites. Pour des environnements de taille complexe, les algorithmes utilisés pour la résolution des PDM ont un temps de calcul très important, ce qui les rend difficilement adaptables dans le cadre d'une application temps-réel. De ce fait, les différents travaux actuels s'intéressent aux techniques permettant d'obtenir plus rapidement des politiques sous-optimales mais néanmoins intéressantes.

Par ce chapitre, nous avons montré que la planification est opérationnelle. Le suivant, présente les caractéristiques du robot utilisé et les choix effectués pour l'exécution réelle de cette planification.

4. Exécution en réel

4.1. Introduction

Dans un premier temps, nous allons décrire le robot que nous allons utiliser pour les expérimentations. Nous définirons ses propriétés techniques et nous examinerons ses capteurs de proximité. Puis, nous détaillerons la technique de localisation que nous avons appliquée à notre problème. Nous expliciterons l'adaptation de la fonction d'observation pour l'exécution réelle. Ensuite, nous exposerons le cadre d'un environnement réel et développerons l'utilisation du robot en pratique. Nous présenterons sa communication, la définition du déplacement et l'abstraction des données fournies par les capteurs. Enfin, nous présenterons une expérimentation de cette exécution en réelle.

4.2. Présentation du robot

4.2.1. Les généralités

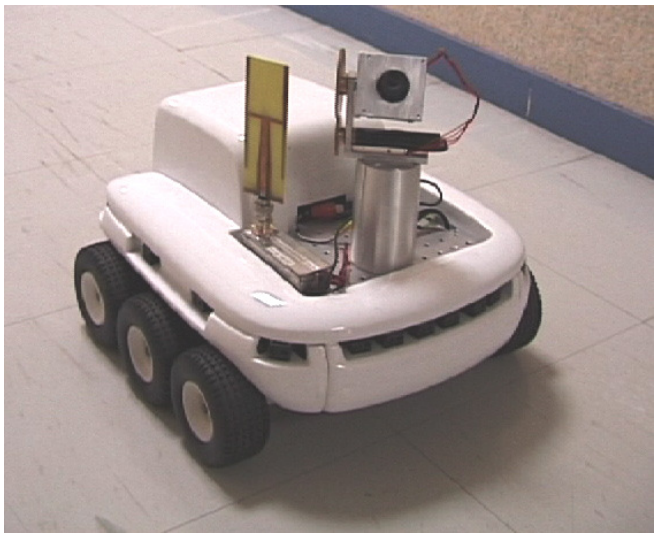


Figure 4.1 *– Photo du Koala.*

Nous allons exposer quelques caractéristiques techniques sur le robot. Tout d'abord, ces dimensions qui seront des paramètres intéressants lors de la définition d'une cellule de l'environnement. Il a une longueur et une largeur identiques de 32 centimètres, pour une hauteur de 20 centimètres. Son poids est d'environ 3 kilogrammes. Son autonomie est définie aux alentours de 3 heures et il se déplace grâce à deux moteurs continus qui gèrent respectivement les mouvements des trois roues situées de chaque côté. Enfin, des pièces essentielles pour sa perception, il est muni de 16 capteurs infrarouges sensibles à la proximité et à la lumière.

4.2.2. Les capteurs infrarouges

Les 16 capteurs infrarouges sont positionnés et numérotés comme représentés dans la figure suivante.

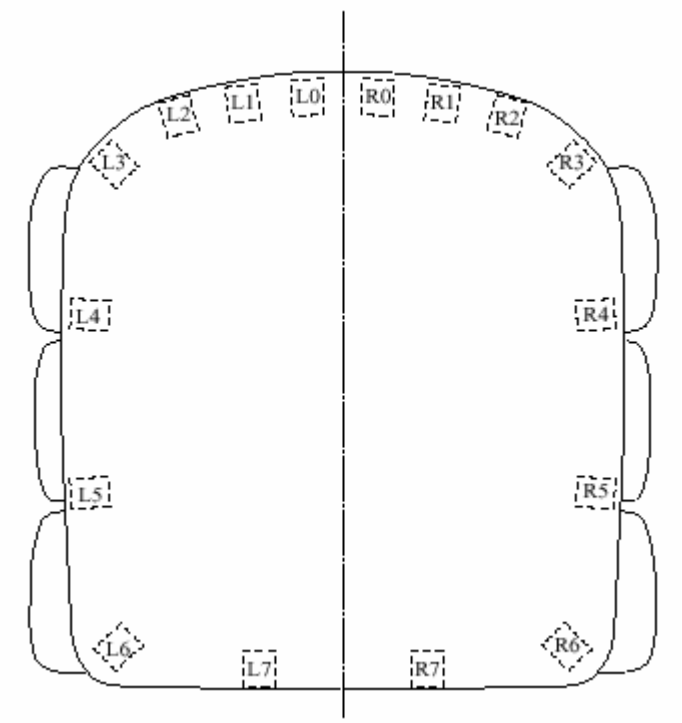


***Figure 4.2** – Position et numérotation des capteurs infrarouges*

Chaque capteur est constitué d'une diode qui émet la lumière infrarouge et d'un récepteur. Ce système permet de quantifier deux éléments : la lumière ambiante et la lumière réfléchie par les obstacles. La mesure de la lumière ambiante est effectuée uniquement par les récepteurs. Pour la proximité, le robot émet une lumière infrarouge et mesure la quantité de lumière reçue. La valeur attribuée à chaque capteur est la différence entre cette mesure et la précédente.

4.2.3. Les mesures

La mesure de la proximité des obstacles ne correspond pas à la mesure d'une distance. Ce type d'évaluation est basé sur la quantité de lumière que l'obstacle réfléchi vers le robot. Cette quantification dépend de deux facteurs : la réflexibilité de l'obstacle (couleur, type de surface, …) et de la lumière ambiante. La figure suivante fournit quelques mesures donnant une idée de la réponse du capteur face à des obstacles de différentes matières et couleurs. Nous remarquons que les différences sont importantes.

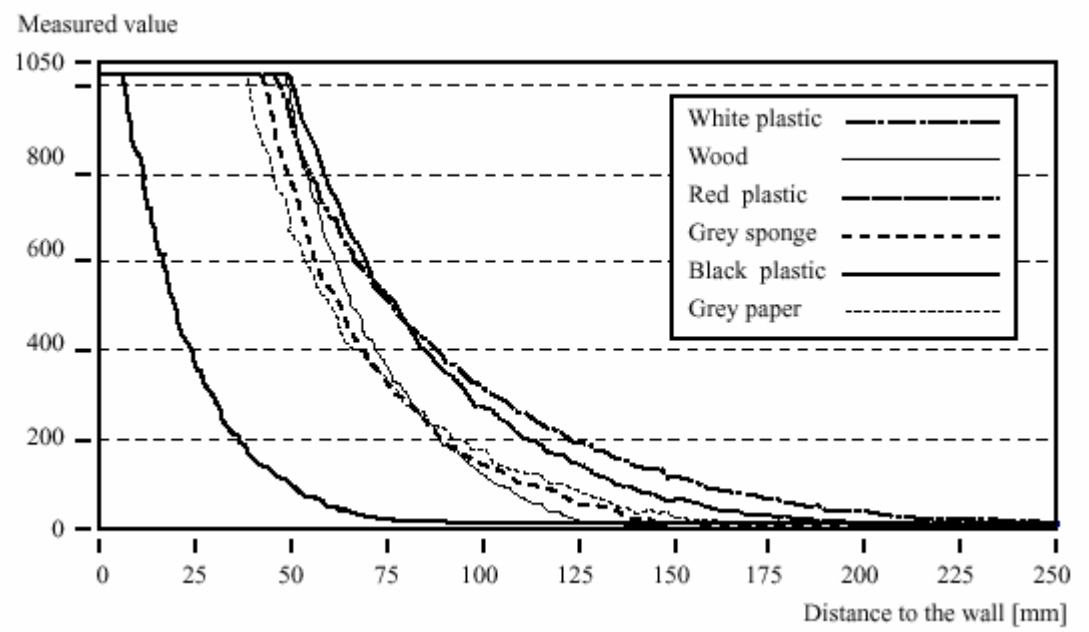


***Figure 4.3** – Mesures suivant différents matériaux et couleurs*

La dépendance de la mesure à la lumière ambiante n'est pas représentée dans la figure précédente. Cependant, elle peut être exprimée ainsi : le récepteur a une sensibilité différente dans des conditions d'éclairage distinct. Si la lumière ambiante est importante alors le récepteur est plus sensible que s'il fonctionnait dans le noir. Pour cette raison, un obstacle détecté dans la lumière retournera une quantité de lumière plus importante que si ce même objet était plongé dans le noir. Donc cette différence peut être importante.

4.3. Observations du robot

Lors d'une exécution en réel, le Koala observe son environnement à l'aide de ses seize capteurs infrarouges. Étant donnée la répartition des capteurs (cf. figure 4.2) sur le robot Koala, nous avons divisé cette « ceinture » de capteurs en cinq zones d'observation (cf. figure 4.4) : une zone frontale composée de six capteurs (L1, L2, L3, R1, R2 et R3), une zone avant gauche et une zone avant droite comportant chacune un capteur (respectivement L4 et R4) et une zone de chaque côté du robot ayant chacune deux capteurs (L5, L6 et R5, R6). Pour chaque zone, nous avons utilisé des observations abstraites de bas niveau. Nous déterminons ainsi à partir des données de proximité des capteurs si la zone est libre ou occupée.

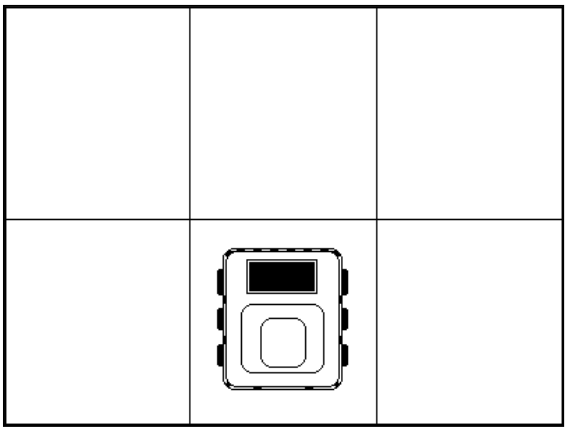


***Figure 4.4** – Zones physiques d'observation utilisées*

Nous avons donc défini un ensemble de 2^5 observations abstraites, chacune étant plus ou moins probables dans les états de l'environnement.

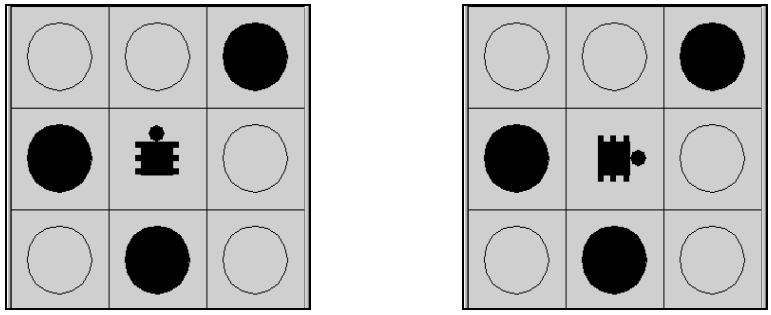
Les zones d'observation correspondent aux états adjacents à la position du robot, il suffit donc pour chaque zone de déterminer s'il s'agit d'un obstacle ou non dans l'environnement.

4.4. Fonction d'observation

Les valeurs des probabilités de chaque état sont mises à jour en fonction des positions probables précédentes, de l'observation du robot et de la fonction de transition. Aussi, la formule précédente (cf paragraphe 3.4.2) fait appel à une fonction d'observation *O* que nous n'avons pas encore défini. Cette fonction renvoie pour chaque couple (*s, o*) la probabilité que le robot fasse l'observation *o* dans l'état *s*. Nous avons choisi d'utiliser par la suite une fonction d'observation qui renvoie pour chaque tuple (s, orient, o) la probabilité que le robot fasse l'observation o dans l'état s et en ayant l'orientation *orient*. Donc notre formule précédente va changer :

$$\Pr_t(s', orient) = \frac{O(s', orient, o) \times \sum_{s \in S} \Pr_{t-1}(s) \times T(s, a, s')}{\sum_{s \in S} \Pr_t(s)}$$

Ainsi, si pendant la planification nous avons pu éliminer l'orientation du robot, ici pour l'exécution elle est inévitable. Les observations du robot dépendent de son orientation. Prenons l'exemple illustré dans la figure 4.5.



a *b*

Figure 4.5 *– Prise en compte de l'orientation du robot*

Si le robot est orienté vers le nord, il aperçoit un obstacle à sa gauche, libre en haut à gauche, libre en face, un obstacle en haut à droite et libre à sa droite (voir figure 4.5 a). Mais par contre si l'orientation du robot était Est, conforme à la figure 4.5 b, les observations du robot seraient : libre en haut (à sa gauche), obstacle en haut à droite (en haut à sa gauche), libre en face, libre en bas à droite (en haut à sa droite) et obstacle en bas (à sa droite). Comme nous venons de le voir, l'orientation du robot est très importante pour la fonction d'observation et pour une bonne exécution d'un plan.

La fonction d'observation *O* a été définie de manière expérimentale étant donnée que pour celle-ci, il n'existe pas de modèle prédéterminé. En fait, pour un tuple (*s, orient, o*), nous comparons ce que le robot observe réellement (en réel), c'est à dire *o* et ce qu'il verrait s'il se trouvait dans l'état *s*, orienté vers *orient* et ceci pour chaque zone. Soit un score compris entre 0 et 5, où le score est en fait le nombre de zones où l'observation *o* et celle dans l'état *s* est la même. Pour chacun de ces scores, nous associons une probabilité, sachant que le score i (où 0 <= i <= 5) correspond à plusieurs combinaisons, nous notons cette probabilité Pr(i / 5). Voici, la fonction d'observation que nous avons implémentée :

- Pr(5 / 5) = 0.5, nombre de combinaison : 1
- Pr(4 / 5) = 0.05, nombre de combinaison : 5
- Pr(3 / 5) = 0.02, nombre de combinaison : 10
- Pr(2 / 5) = 0.004, nombre de combinaison : 10
- Pr(1 / 5) = 0.0019, nombre de combinaison : 5
- Pr(0 / 5) = 0.0005, nombre de combinaison : 1

Là encore, nous avons intégré la possibilité pour l'utilisateur de modifier cette fonction, mais les nouvelles probabilités devront respecter la propriété suivante :

$$\sum_{i=0}^{5} \Pr(i/5) \times C_i^5 = 1$$

4.5. Environnement réel

Lors des futures expérimentations, nous devrons définir un environnement réel. Pour ce faire, nous avons à notre disposition des planches en formica blanc qui nous permettront de délimiter le terrain et de simuler des obstacles immobiles (murs, cloisons, …).

Cependant, cette mise en place comporte plusieurs contraintes. Nous devons prévoir une superficie d'implémentation non négligeable qui sera monopolisée pendant plusieurs jours. Le nombre et la distance globale des planches sont limités donc nous devons envisager un environnement réalisable. Enfin, nous devons déterminer la surface adéquate pour définir une case de l'environnement en fonction du gabarit du robot et de ses capacités de perception.

Pour ce dernier point, nous avons fixé les côtes comme représentées dans la figure suivante.

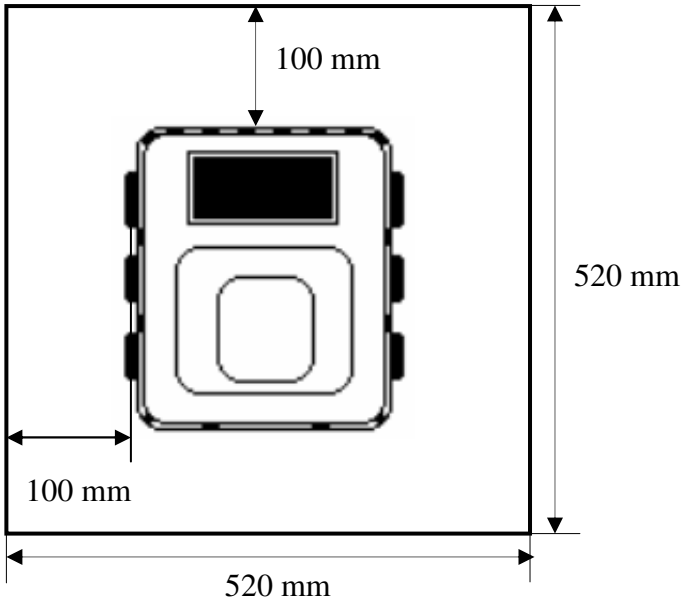


***Figure 4.6** – Définition d'une case de l'environnement*

Nous pouvons remarquer que la case est un carré de 520 millimètres de côté. Cette côte est approximative. En fait, nous pouvons accepter une erreur de ± 5%. Lorsque le robot est situé au centre de la case alors il est à une distance respective de 100 millimètres des côtés de la cellule. Cet espacement semble important au vue de la capacité des capteurs infrarouges. Cependant, il assure au robot une marge lui permettant de pivoter sur lui-même sans entrer en collision avec les obstacles.

Cette définition est spécifique à notre cas de figure puisque nous utilisons des obstacles dont les propriétés, surtout la couleur blanche, sont fortement réfléchissantes. Donc ce choix sera à remettre en cause si nous modifions ces particularités.

4.6. Utilisation du robot

Lors de l'exécution, nous avons deux acteurs principaux : le robot et le logiciel. L'application informatique nous permettra de définir une carte de l'environnement. Elle élaborera la planification associée. Elle calculera la distribution de probabilités à partir des informations fournies par les capteurs qui lui permettront de réaliser la localisation du robot sur la carte. Elle déterminera également les actions que le robot devra effectuer et les lui fournira. Enfin, elle affichera sous la forme d'une interface graphique ces différents renseignements.

De son côté, le robot exécutera les ordres envoyés et livrera les données issues de ses capteurs. De plus, il devra éviter les obstacles définis sur la carte de son environnement et ceux qui pourraient survenir.

Afin que chaque protagoniste réalise les tâches qui lui sont adjugées et échange les informations nécessaires à leurs déroulements, ils devront s'entretenir au travers d'une communication.

4.6.1. La communication

Pour dialoguer, le robot et l'application informatique doivent établir une communication. Cet entretien s'effectue à partir de différents éléments indispensables pour assurer la conformité des échanges suivant les caractéristiques de chacun. Et nous devons garantir cette communication malgré les divergences existantes.

Suite à divers choix techniques, Adriana TAPUS a programmé le logiciel en langage JAVA. Or, la communication avec le robot Koala (envoi d'ordres, perception des capteurs) doit s'effectuer en langage C++. Donc nous devons établir une passerelle entre les deux interlocuteurs. Pour effectuer cette correspondance, nous avons fabriqué une librairie dynamique et échangé les informations par un système de fichiers. La figure suivante schématise cette communication.

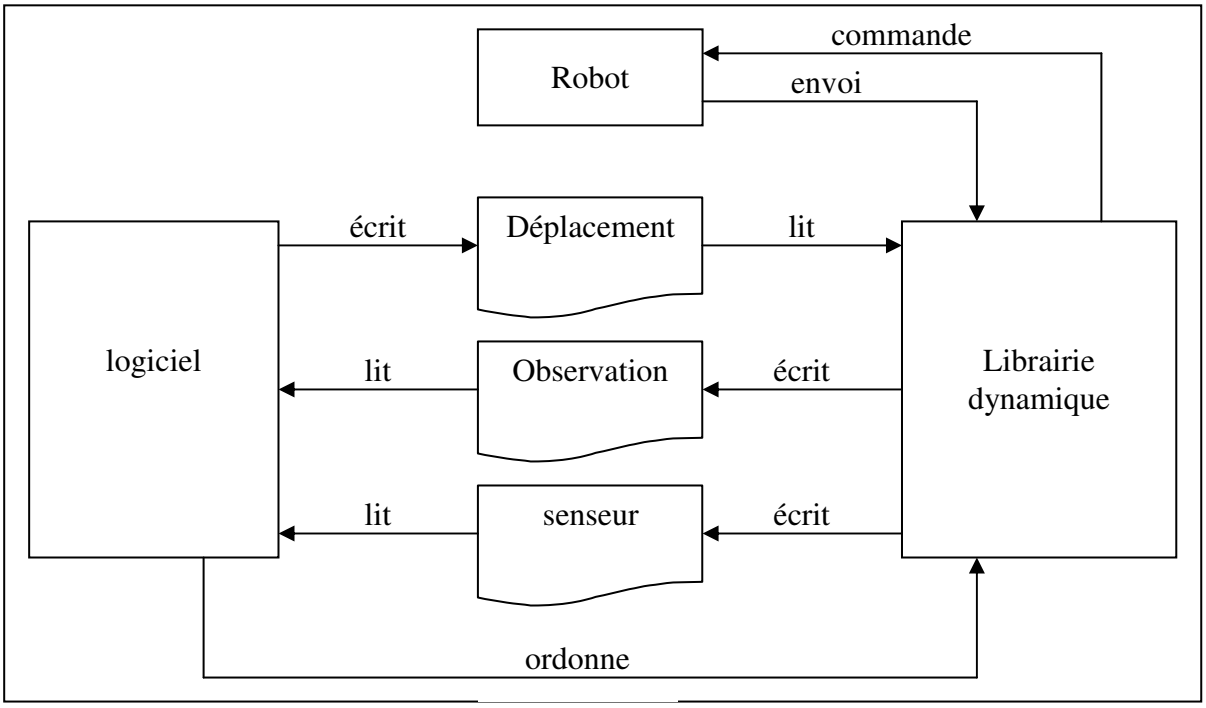


***Figure 4.7** – Schéma de communication*

Remarque : Nous avons utilisé un nouvel élément appelé « senseur ». C'est un capteur qui permet de définir une distance relative et une direction entre deux points (exemple : balise Argos).

Nous allons exposer plus précisément le fonctionnement et l'interaction entre les différents éléments grâce aux deux scénarii suivant.

Pour notre premier scénario, nous supposons que le logiciel souhaite réaliser une observation. Alors, il ordonne à la librairie dynamique de lui fournir les mesures des capteurs de proximité du robot. La librairie dynamique commande au robot de réaliser cette mesure. Il lui envoie les données fournies par les capteurs. Puis, elle les écrit dans le fichier nommé « observation ». Enfin, le logiciel les lit et les traite pour obtenir une observation discrétisée. Ce traitement sera explicité dans la suite du chapitre (cf paragraphe 4.6.3).

Nous notons que la procédure est identique pour effectuer une mesure sur le senseur.

Dans notre second scénario, nous admettons que le logiciel veut que le robot se déplace. Alors, il définit un déplacement (le formalisme des déplacements est explicité dans le paragraphe 4.6.2) et l'écrit dans le fichier nommé « Déplacement ». Puis, il ordonne à la librairie dynamique d'effectuer ce déplacement. La librairie dynamique lit la définition du mouvement dans le fichier et donne des instructions au robot afin de réaliser le déplacement. Ensuite, elle lui commande d'effectuer une mesure des capteurs de proximité. Il lui envoie les données fournies par les capteurs. Alors, elle les écrit dans le fichier « observation » (elle réalise de même pour le senseur si besoin). Finalement, le logiciel lit les données inscrites dans le fichier et les traite pour obtenir une observation discrétisée.

Maintenant, nous allons décrire le formalisme utilisé par le logiciel pour exécuter un déplacement.

4.6.2. Le déplacement

Nous avons indiqué précédemment que la phase de planification ne tenait pas compte de l'orientation du robot. Or, lorsque nous exécuterons ce plan, nous devrons connaître cette orientation afin de réaliser le déplacement souhaité.

Pendant la simulation, l'application informe le robot des mouvements à effectuer : « AVANCER_N », « AVANCER_S », AVANCER_E », « AVANCER_W ». Ce type de commande doit être adapté au langage du robot. Il ne connaît en aucune manière la direction des points cardinaux. Il possède uniquement leur agencement : s'il est orienté au nord alors l'est est à droite, l'ouest à gauche et le sud vers l'arrière. Donc nous devons retranscrire ces ordres. Enfin, il dispose des facultés d'avancer et de pivoter.

Pour résoudre son incapacité à situer les directions cardinales, nous avons défini deux informations. La première nous permet de lui définir l'orientation suivant laquelle il est et la seconde l'orientation suivant laquelle il doit se diriger. Connaissant ces deux renseignements et l'agencement des points cardinaux, il peut se mouvoir. Par conséquent, le logiciel doit inscrire dans le fichier « Déplacement » : l'orientation de la position supposée du robot au départ du mouvement et le sens de déplacement défini par la planification pour cette position supposée.

La seconde information nous est fournie par la planification de l'environnement. Nous illustrons cette affirmation sur la figure suivante.

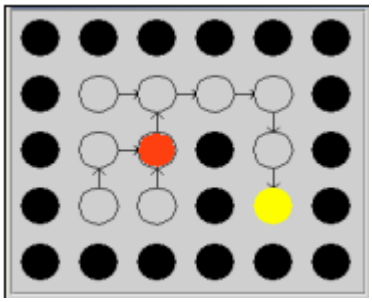


Figure 4.8 – *Sens de déplacement*

Le rond rouge  indique la position supposée du robot. Nous observons que pour cette position et quelque soit l'orientation, la planification nous indique un sens de déplacement vers le nord. Donc la seconde information sera « Nord ».

Pour déterminer la première information, nous comparons l'observation réelle du robot et les observations suivant les quatre directions cardinales. Celle qui obtient la plus forte probabilité est retenue et nous permet de déterminer une orientation du robot. Nous illustrons ce procédé sur la figure suivante.

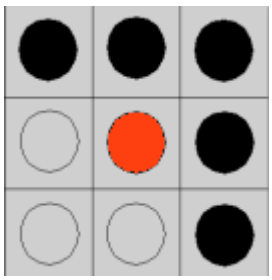


Figure 4.9 *– Orientation de la position supposée du robot*

Le rond rouge  indique la position supposée du robot. S'il est orienté au nord alors son observation devrait être : des obstacles de tous les côtés excepté à gauche. S'il est orienté au sud alors son observation serait : aucun obstacle excepté à gauche et en avant à gauche. S'il est orienté à l'est alors son observation devrait être : des obstacles de tous les côtés excepté à droite. Enfin, S'il est orienté à l'ouest alors son observation serait : aucun obstacle excepté à droite et en avant à droite.

Maintenant, nous supposons que l'observation réelle faite par le robot est définie par des obstacles de tous les côtés exceptés à droite. L'orientation du robot la plus probable pour cette position supposée serait l'est. Donc la première information sera « Est». Par conséquent, le logiciel devra inscrire dans le fichier « Déplacement » l'information « Est-Nord ».

Lorsque deux observations ont la même probabilité de correspondance avec la réelle, nous conservons celle qui maintient le cap du robot. Pour illustrer ce propos, nous étudions l'exemple du couloir présenté dans la figure suivante.

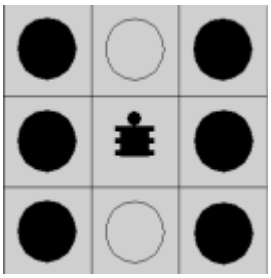


Figure 4.10 – *Cas du couloir*

Nous remarquons que si le robot, représenté par  , est orienté au nord ou au sud alors l'observation pour cette position supposée est identique. Donc nous avons deux correspondances possibles pour une observation réelle définit par des obstacles de tous les côtés excepté devant. Or, si auparavant, le robot se déplaçait vers le nord alors nous conserverons cette orientation pour la direction supposée du robot. Dans ce cas, la seconde information serait « Nord ».

De manière plus technique, nous avons défini quatre actions physiques élémentaires : avancer d'une case, faire un demi-tour, faire un quart de tour à gauche et faire un quart de tour à droite. Ainsi, nous décomposons les deux informations de mouvement inscrites dans le fichier « Déplacement » en une série d'actions élémentaires. Par exemple, pour une donnée « Nord-Est », nous effectuons un quart de tour à droite et avançons d'une case. Nous notons que l'action avancer d'une case dépend fondamentalement des dimensions d'une cellule donc nous devons prendre garde à toute modification de ces paramètres.

Lors de son déplacement, le robot contrôle en permanence la présence d'obstacle sur son cheminement. Suivant leur position, soit il les évite, soit il s'arrête afin de parer toute éventualité de collision. Nous allons illustrer ces deux cas dans les figures suivantes.

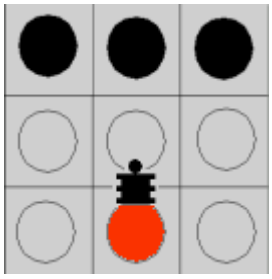


Figure 4.11 – *Arrêt face à un obstacle*

Supposons que suite à de nombreuses incertitudes, la position réelle du robot soit représentée par l'icône noire  et que sa position supposée soit définie par le rond rouge  . L'information contenue dans le fichier « Déplacement » lui définit comme déplacement « Nord-Nord ». Donc le robot doit avancer d'une case. Afin d'exécuter sa mission, le robot avance tout droit et s'arrête à quelques centimètres de l'obstacle. Un avantage supplémentaire de cet évitement est sa faculté à recentrer le robot par rapport à une case et de diminuer l'incertitude.

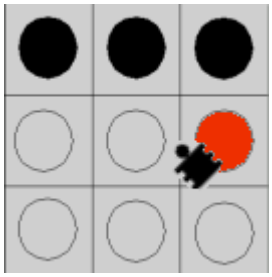


Figure 4.12 – *Evitement d' un obstacle*

Admettons que suite à de nombreuses incertitudes, la position réelle du robot soit représentée par l'icône noire  et que sa position supposée soit représentée par le rond rouge  . L'information contenue dans le fichier « Déplacement » lui définit comme déplacement « Nord-Nord ». Donc le robot doit avancer d'une case. Afin d'exécuter sa mission, le robot avance tout droit et poursuit son déplacement en longeant l'obstacle par la droite pour l'éviter. Un avantage supplémentaire de cet évitement est sa faculté à remettre le robot dans l'axe et de diminuer l'incertitude.

Après chaque déplacement, le logiciel demande au robot les valeurs de ces douzes capteurs. Puis, il les examine et les abstrait. Dans le paragraphe suivant, nous présentons la réalisation de cette abstraction.

4.6.3. L'observation

Cette phase de l'exécution demeure très expérimentale. Puisque nous devons abstraire les données fournies par les douze capteurs infrarouges en cinq informations spécifiant la présence ou l'absence d'obstacles suivant des zones (cf paragraphe 4.3).

Pour la suite de l'explication, nous rappelons que plus un obstacle est près d'un capteur plus la valeur mesurée est grande (cf paragraphe 4.2.3). Pour réaliser ce traitement, nous avons simplement défini des seuils adaptés à la position des capteurs. Puis, nous appliquons la règle suivante : si la plus grande valeur mesurée sur les capteurs associés à une zone est supérieure au seuil fixé pour cette même zone alors nous sommes en présence d'un obstacle pour cette zone, sinon elle est libre.

Nous illustrons cette règle avec l'exemple suivant. Pour la zone frontale, nous examinons la valeur des capteurs L1, L2, L3, R1, R2 et R3. Nous conservons la plus élevée et nous la comparons au seuil défini pour cette zone. Si elle est supérieure au seuil alors cette zone est un obstacle, sinon elle est libre. Ensuite, nous faisons de même pour les quatre autres zones suivant les capteurs et les seuils affiliés.

Nous notons que la détermination de ces seuils est étroitement liée à l'éclairage global du site expérimental et aux propriétés des éléments le définissant (propriétés réflexives des éléments définissant l'environnement).

Nous venons de définir tous les composants nécessaires pour l'exécution réelle d'une planification. Donc, nous allons présenter dans le paragraphe suivant une expérimentation de cette exécution.

4.7. Expérimentation

Nous allons exposer une expérimentation simple où nous connaissons la position du but mais la position initiale du robot est inconnue. Puis, nous présenterons un problème que nous avons rencontré. Pour la suite de ce paragraphe nous définissons plusieurs icônes :

-  position du but.
-  position supposée du robot.
-  position réelle du robot.
-  obstacle.
-  cellule libre.

Nous allons également introduire un formalisme : lorsque nous donnerons le résultat d'une observation, nous donnerons 0 si libre et 1 s'il y a un obstacle et nous fournirons dans l'ordre suivant l'état de la zone à gauche (cf figure 4.4), de la zone avant gauche, de la zone frontale, de la zone avant droite et de la zone droite. Par exemple, une observation où nous avons des obstacles de tous les côtés excepté devant donnera : 11011.

La figure 4.13 représente la carte de l'environnement que nous avons utilisé et la figure 4.14 sa planification.

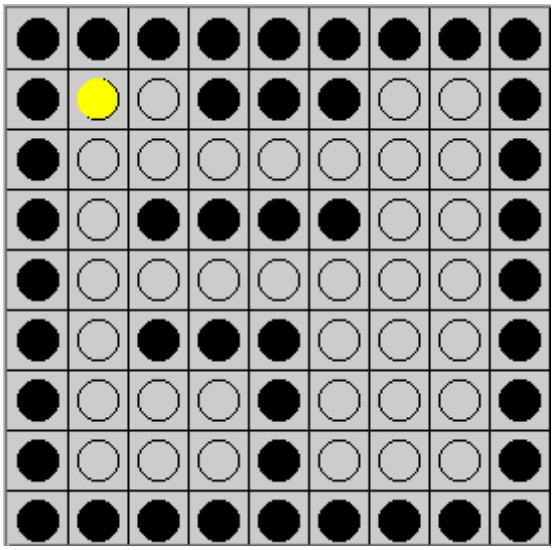


Figure 4.13 – *Carte de l'environnement*

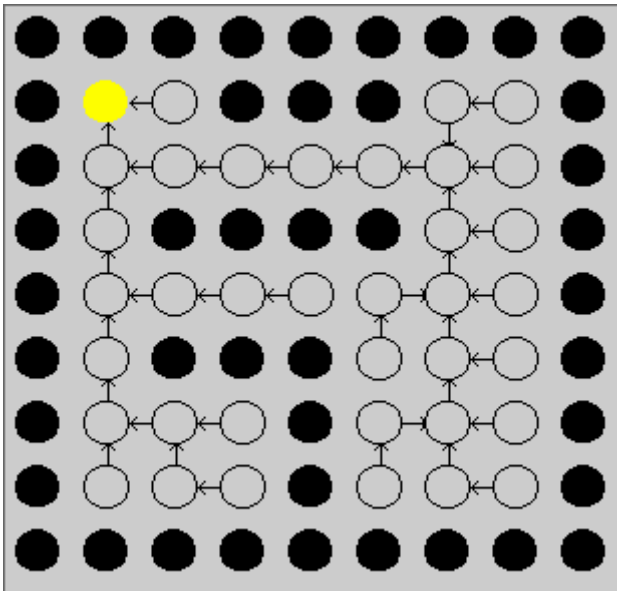


Figure 4.14 *– Planification de la carte*

Chacune des figures qui suivent est composée d'un cadre supérieur qui expose la distribution de probabilités en sachant que plus la teinte est foncée plus la probabilité est forte et inversement. Le cadre inférieur donne la position supposée du robot et la position du but. Nous avons ajouté par un artifice graphique la position réelle du robot.

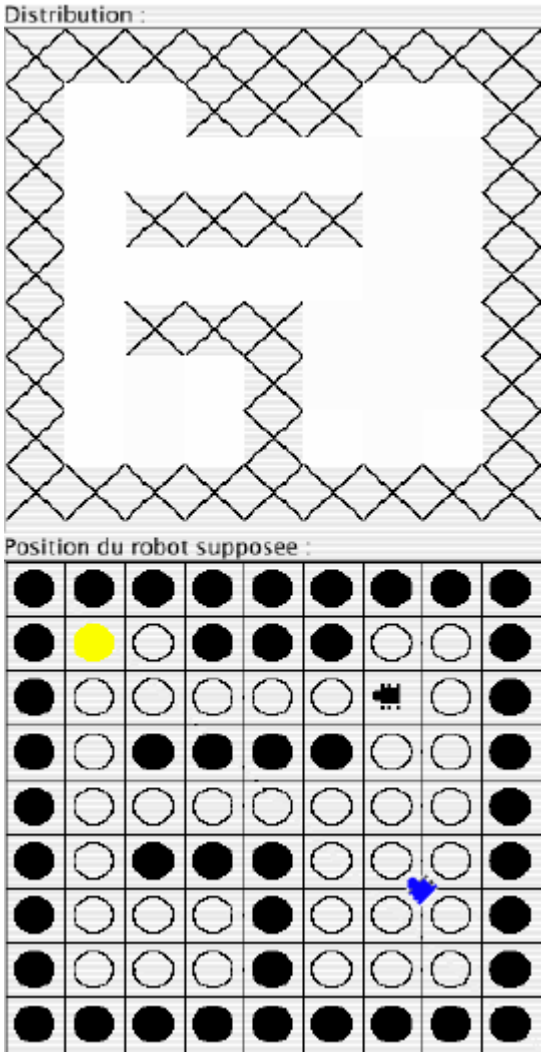


Figure 4.15 – *Première étape*

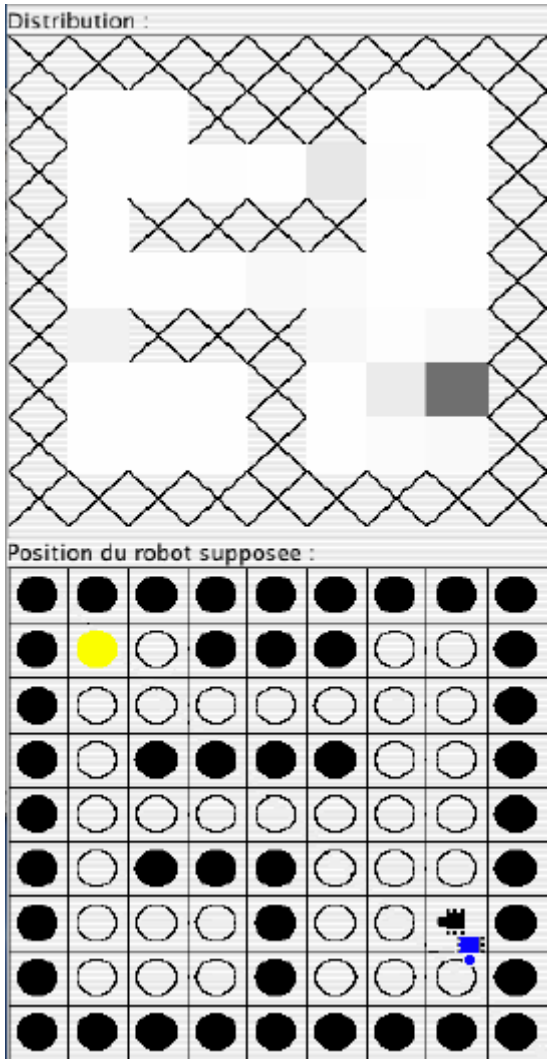


Figure 4.16 *– Première étape*

Cette première étape consiste en une simple observation qui donne 00000. La couleur claire de la distribution est due aux faibles valeurs des probabilités.

Puis, nous effectuons le déplacement « Est-Ouest » et nous obtenons l'observation 11000. Nous remarquons que la localisation de la position supposée a été très rapide.

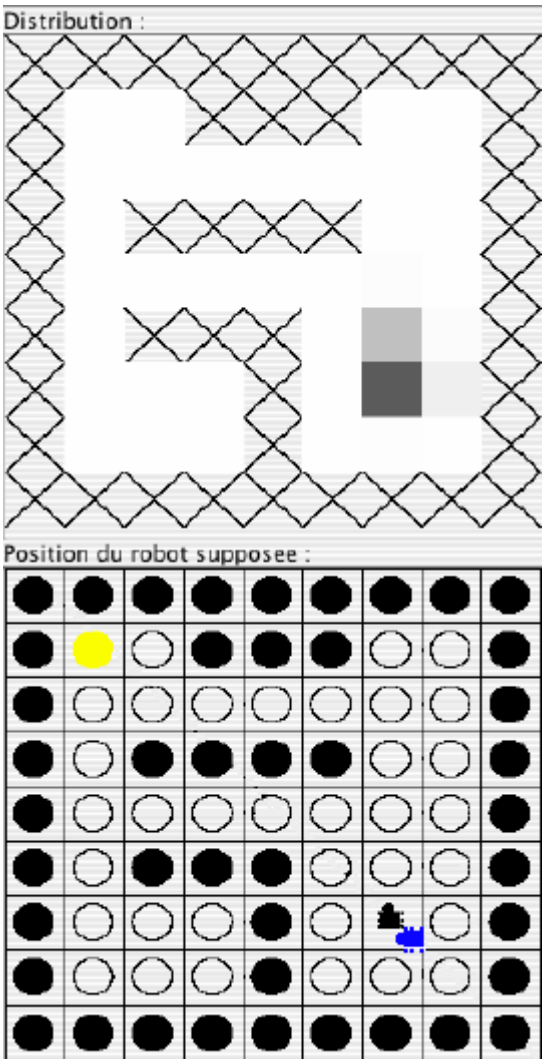


Figure 4.17 – *Troisième étape*

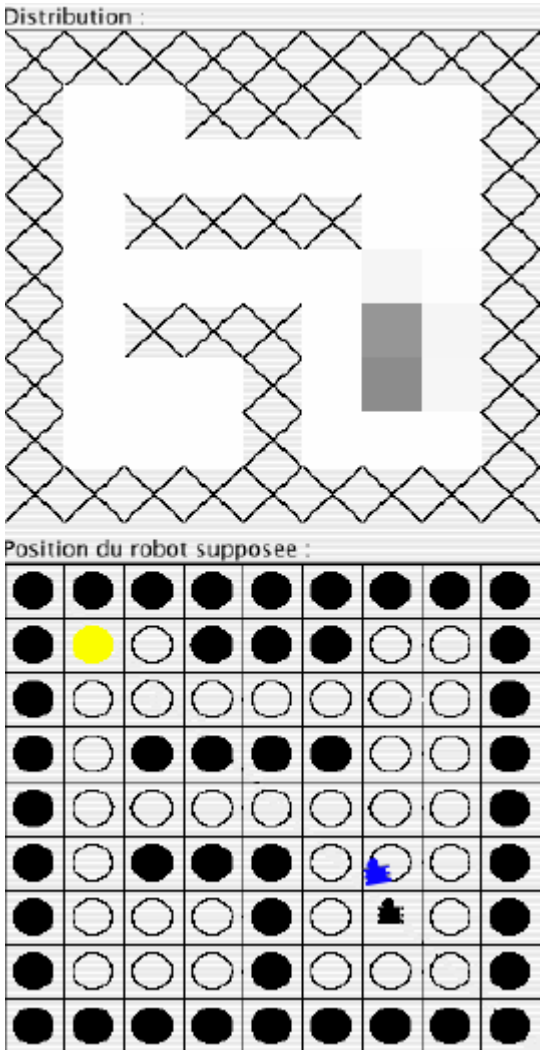


Figure 4.18 – *Quatrième étape*

Puis, nous effectuons le déplacement « Sud-Ouest » et nous obtenons l'observation 00000. Nous remarquons que la probabilité se disperse puisque nous avons une observation non-discriminante.

Puis, nous effectuons le déplacement « Ouest-Nord » et nous obtenons l'observation 00000. Nous remarquons que la probabilité se concentre puisque nous avons effectué trois actions qui ont donné la même observation donc très driscriminantes.

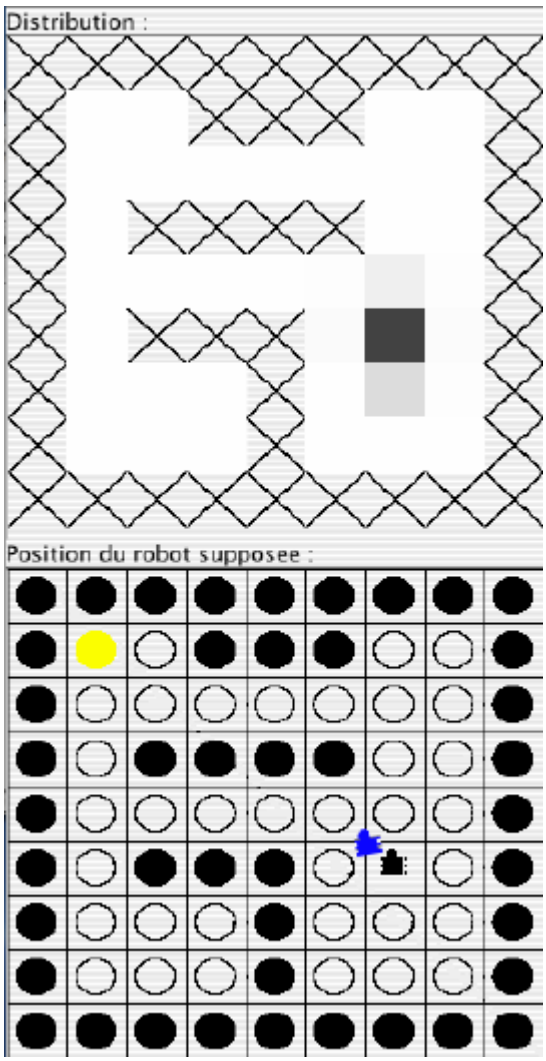


Figure 4.19 – *Cinquième étape*

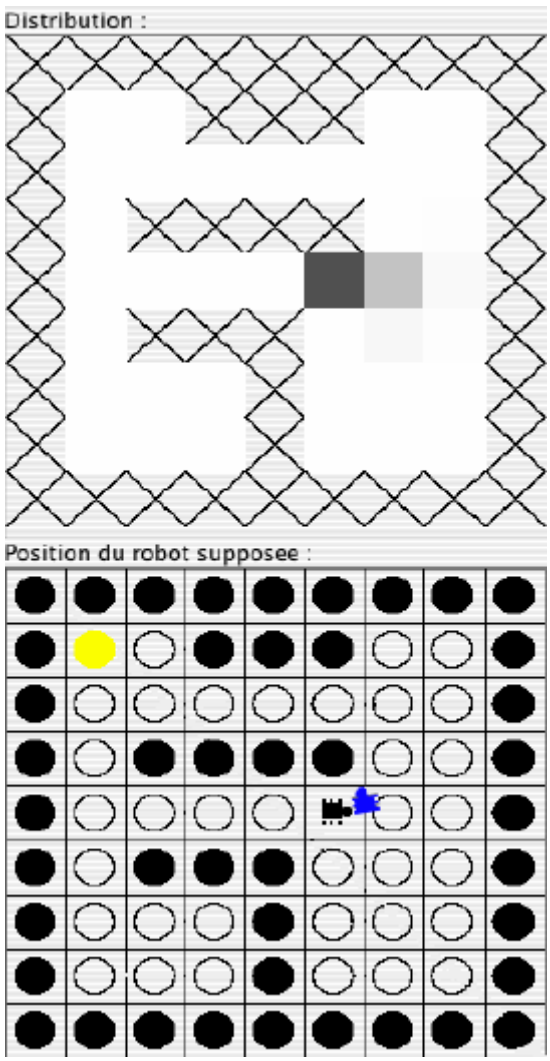


Figure 4.20 *– Sixième étape*

Puis, nous effectuons le déplacement « Nord-Nord » et nous obtenons l'observation 01100. Cette observation est très discriminante et conserve la concentration de la probabilité.

Puis, nous effectuons le déplacement « Nord-Nord » et nous obtenons l'observation 00000. La probabilité se diffuse légèrement due aux incertitudes de déplacements.

Nous remarquons dans l'ensemble des expérimentations que nous avons faites que le modèle et la détermination des paramètres sont bien adaptés à ce type de problèmes. Nous localisons assez rapidement le robot avec une marge d'erreurs faible. Lorsque ces erreurs surviennent : bruits sur les capteurs, incertitude de déplacements, … Nous retrouvons aisément la position du robot en quelques déplacements.

Cependant, dans le cas où le robot se situe dans la zone la plus vaste comme signalé par un carré rouge sur la figure 4.21. Le nombre de déplacements et d'observations s'accroît conséquemment pour localiser la position du robot. Cette zone ayant peu de caractéristiques discriminantes. Mais sa localisation aboutit au bout de plusieurs dizaines d'actions.

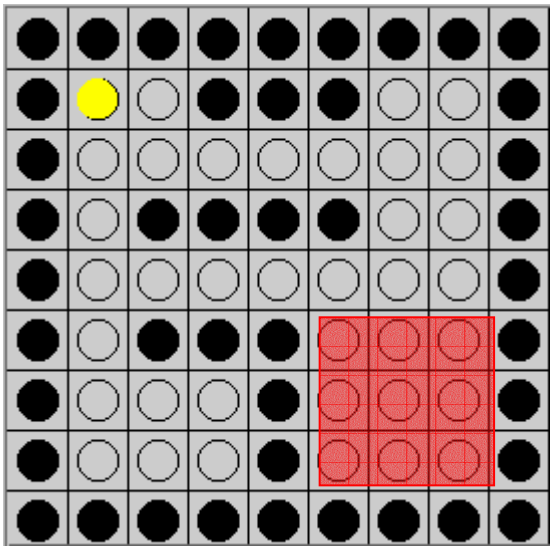


Figure 4.21 – *Zone peu discriminante*

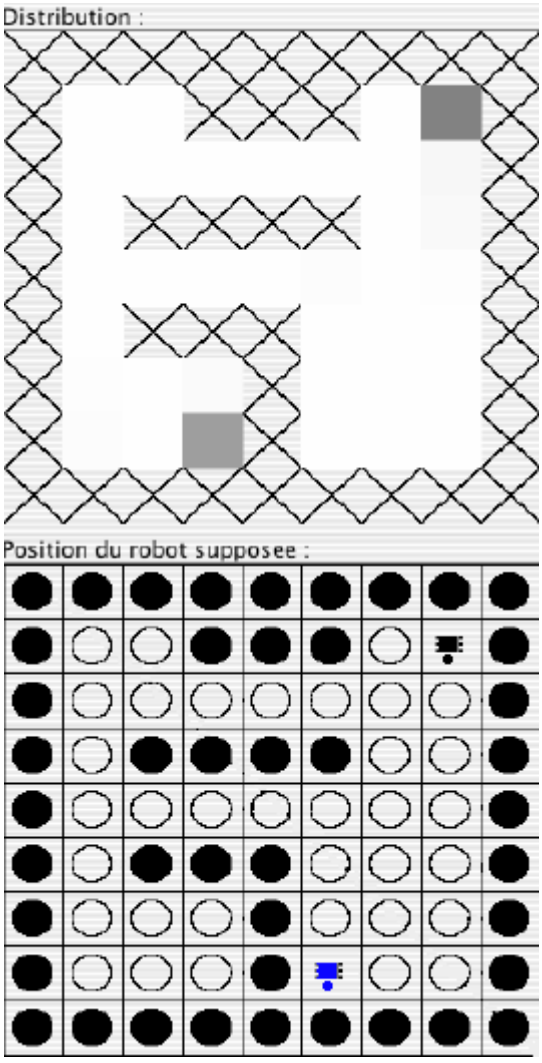


Figure 4.22 – *Problème insoluble*

Toutefois, un problème insoluble nous est apparu. Lorsque certaine partie de l'environnement sont trop similaire. Le logiciel pense que d'après ses caractéristiques, le robot a atteint la position du but mais que la probabilité de cette position n'est pas assez importante pour terminer le processus. Le logiciel du fait de la similarité des incertitudes n'arrive pas à conclure l'expérimentation et a une importante erreur sur la position supposée du robot. Le logiciel effectue alors un cycle qui définit toujours les mêmes probabilités et ramène le robot toujours au même endroit. Tel est le cas dans la figure ci-contre.

Nous avons qualifié ce problème d'insoluble car le nombre important d'actions que nous avons effectué n'a pas permis au logiciel de sortir de cette impasse. Mais sur un nombre d'actions qui tend vers l'infini, l'ampleur des incertitudes grandissantes devrait remédier au problème à long terme.

Un moyen plus rapide pour échapper à ce problème est l'utilisation d'un capteur supplémentaire. Dans notre cas le senseur qui discriminera immédiatement les différents cas possibles.

4.8. Conclusion

Comme nous l'avons vu, les observations faites par le robot l'aident à se localiser. En comparant l'observation courante aux observations possibles à partir des divers états, nous pouvons déterminer un sous-ensemble d'états dans lequel le robot a une probabilité forte de se trouver. En conséquence, plus le nombre d'observations considérées est grand, plus le robot pourra se localiser correctement. Les observations permettent également d'éviter des collisions pouvant endommager le robot.

L'utilisation des *belief states* permet d'agir en n'ayant qu'une connaissance partielle de l'état courant, la localisation se faisant à l'aide des facultés de perception du robot. Si le plan est connu à l'avance, la présence d'obstacles imprévus n'est pas dangereuse pour l'exécution du plan puisque la localisation se fait en fonction des observations faites. Si le robot observe un obstacle imprévu, sa croyance quant à l'état courant sera faussée, mais l'action effectuée sera choisie en fonction de l'observation courante et donc les chocs seront évités. Bien sûr, la présence de nombreux obstacles imprévus rendra le plan inutilisable.

Cette localisation dépend profondément des capacités de perception du robot. Or, la qualité des informations fournies est directement liée à la détermination des paramètres du robot et aux propriétés des éléments qui composent l'environnement. Donc leur adéquation demeure essentielle pour la réalisation des expériences.

Dans la session suivante, nous allons complexifier le problème. Lors des expérimentations réalisées dans ce chapitre, nous connaissions avec certitude la position du but mais la position initiale du robot était inconnue. Nous devions seulement localiser le robot. Dans le cas suivant, nous disposons seulement de deux informations : la position relative du but par rapport au robot et les observations faites par le robot.

5. Recherche et poursuite d'une cible mobile

5.1. Introduction

Dans ce dernier chapitre, nous présentons une application des Processus Décisionnels de Markov à un problème plus complexe. Il s'agit pour le robot de rechercher et de poursuivre un but mobile dont la position est inconnue. De plus, la position initiale du robot est inconnue et la seule information dont il dispose sur le but à atteindre est sa position relative par rapport à celui-ci. Nous pouvons comparer ce contexte à un prédateur poursuivant sa proie.

Dans la section suivante, nous présentons la méthode que nous avons utilisé pour résoudre ce problème et ensuite nous illustrons la technique sur une expérimentation.

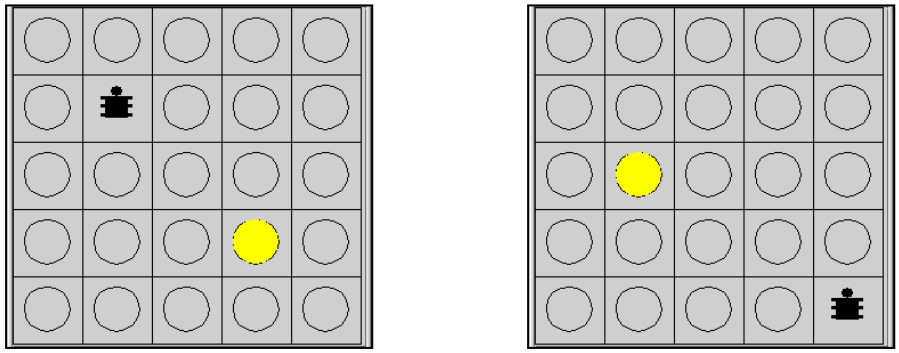
5.2. Description du problème

La cible et le robot évoluent dans un environnement connu et défini à l'aide de l'application. La carte de l'environnement décrit les dimensions à partir d'unités définies par les cases. Elle établit également la position des obstacles immobiles.

Les positions initiales du robot et de la cible sont totalement inconnues. Seule la position relative de l'un par rapport à l'autre est connue. De plus, la cible effectue une stratégie de déplacement au cours de l'exécution. Dans notre cas, nous considérons la cible comme une proie donc elle essaiera d'échapper à son poursuivant. Cependant, elle aura une vélocité moins importante que son prédateur. Cette clause est primordiale si nous voulons que le robot attrape cette cible.

Toutes les informations dont nous disposons sont les observations du robot et quelques données qui sont transmises par les senseurs du robot. Ces données nous disent à combien de cases se trouve le but par rapport au robot.

Soit la figure 5.1.



a *b*

Figure 5.1 *– Positionnement du but par rapport au robot*

Si nous regardons la figure 5.1a, nous pouvons apercevoir que le but se trouve par rapport au robot à 2 lignes et à 2 colonnes plus loin. Nous pouvons avoir aussi des valeurs négatives, comme le montre la figure 5.1b. Le but se trouve à -2 lignes et à -3 colonnes par rapport au robot.

Pour généraliser, nous avons mis en place un repère. L'origine est la position supposée du robot. L'axe des abscisses est défini suivant la verticale et du haut vers le bas. L'axe des ordonnées a pour direction l'horizontale et son sens est de la gauche vers la droite. Donc sa valeur est un nombre de lignes (positif ou négatif). Ce repère est représenté sur la figure suivante.

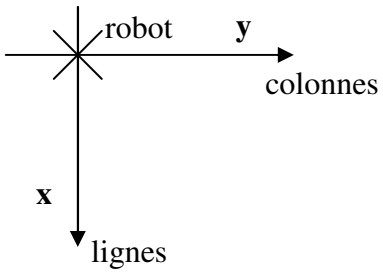


***Figure 5.2** – Repère pour la localisation du but par rapport au robot*

Au début, nous ne savons pas où se trouve le but. Comme nous l'avons dit précédemment, la seule information concernant le but sont les données du senseur du robot. Donc, nous allons mettre sur la carte le but le plus probable qui dépend de l'hypothèse sur la position du robot. Cette hypothèse sera l'état dans lequel le robot est supposé se trouver.

Au départ, la position initiale du robot est inconnue et donc nous allons avoir une distribution uniforme. Chaque fois que le robot fait de nouvelles observations, une nouvelle distribution va être calculée. Donc nous pourrons déterminer la position supposée du robot et par déduction celle du but.

5.3. Méthode de résolution

5.3.1. Exposé de la méthode

La méthode que nous avons employée se décompose en quatre phases :

- La localisation : nous déterminons en fonction de l'observation faite par le robot sa position supposée la plus probable dans l'environnement.
- La formulation d'hypothèses : Après avoir identifier cette position, nous allons émettre une hypothèse sur la position du but. Nous avons à notre disposition la distance relative de la cible par rapport au robot. Donc en l'appliquant sur la position supposée du robot, nous émettons une position supposée du but.

- La planification : nous connaissons la position supposée de la cible donc nous pouvons réaliser la planification de la carte de l'environnement.
- L'exécution : nous avons, en notre possession, une carte planifiée et la position supposée du robot. Donc nous pouvons définir un déplacement pour l'exécuter sur le robot. Suite à cette dernière phase, nous retournons sur la première et nous bouclons tant que nous n'avons pas attrapé la proie.

Finalement, nous allons faire le calcul du plan et son exécution en parallèle. Une fois qu'un premier plan est calculé, nous pouvons commencer à l'exécuter, pendant que le processus l'améliore. A chaque fois qu'un nouveau plan est trouvé, il est envoyé au processus d'exécution qui abandonne aussitôt le précédent. Ceci permet au robot d'agir avant que le plan optimal soit calculé et d'atteindre le but plus rapidement.

Nous pouvons faire une telle technique, c'est-à-dire de paralléliser les deux méthodes : la planification et la localisation, car elles utilisent des algorithmes qui effectuent un calcul itératif.

Dans ce paragraphe, nous décrivons plus précisément le déroulement de la méthode. Dans un premier temps, nous effectuons une observation qui va nous permettre d'émettre des premières hypothèses sur la position supposée du robot. Grâce à l'information issue du senseur nous connaissons la position relative du but par rapport au robot. Cette donnée nous permet de trier les hypothèses faites. Ainsi, nous défavoriserons les hypothèses improbables au vue de la position relative de la proie et favoriserons les autres. Par exemple, nous supposons le robot à la position (x, y) et le senseur nous fournit les données (x_s, y_s). Si la coordonnée (x+x_s, y+y_s) est un obstacle ou sort du cadre de l'environnement alors nous dévaluerons la probabilité pour la position (x, y) sinon nous la conforterons. Nous effectuons des diminutions sur les probabilités mais nous ne les éliminons pas car les informations fournies par le senseur ne sont pas parfaites. Ainsi, nous conservons des hypothèses peu probables mais non négligeables. Nous illustrons ce procédé dans les figures suivantes. Chacune des figures est composée d'un cadre supérieur qui expose la distribution de probabilités et celui inférieur donne la position supposée du robot.

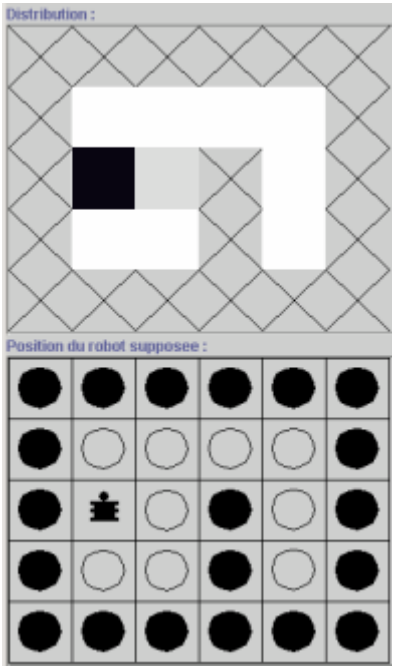


Figure 5.3 – *Etat avant correction*

Nous supposons que le robot est à la position (2, 1), représenté par l'icône noire . Le senseur nous fournit les données (1, 2). Si nous ajoutons aux coordonnées du robot la distance relative fournie par le capteur, nous obtenons la position supposée (3, 3) pour le but. Or, elle définit un obstacle. Donc nous allons dévaloriser la valeur de la probabilité de la position supposée du robot. Comme nous pouvons le constater, la teinte foncée, spécifiant une forte probabilité, de la position (2 , 1) de la figure 5.3 est beaucoup plus claire sur la figure 5.4, spécifiant une faible probabilité.

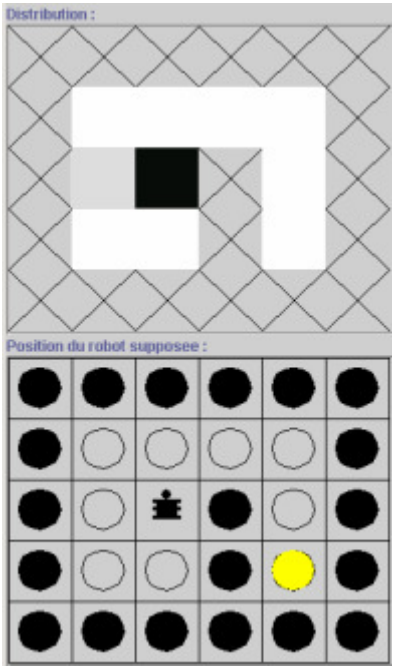


Figure 5.4 *– Etat après correction*

Contrairement au cas de cette position, si nous considérons la position (2, 2), fortement moins probable, et que nous ajoutions la distance relative alors nous obtenons la position supposée (3, 4) pour le but. Or, nous obtenons une hypothèse potentielle. Donc nous allons valoriser la valeur de la probabilité de la position (2, 2). Comme nous pouvons le constater, la teinte claire, spécifiant une faible probabilité, de la position (2 , 2) de la figure 5.3 est beaucoup plus foncée sur la figure 5.4, spécifiant une forte probabilité. Donc cette position devient la nouvelle position supposée du robot et la nouvelle position supposée du but est (3, 4). Elle est représentée par un rond jaune .

Ensuite, nous allons faire une itération de l'algorithme *Policy Iteration* afin d'obtenir une planification partielle de l'environnement. Puis, nous exécutons une action. Enfin, nous retournons une observation et la méthode se poursuit.

Plus la méthode se répète et plus la position supposée du robot aura une forte probabilité. D'où une position de la proie affinée et une planification relativement constante.

5.3.2. Résultats

Nous allons exposer une expérimentation complexe où nous ne connaissons ni la position initiale du robot, ni celle du but. Les seules informations dont nous disposons sont la distance relative du but par rapport au robot et les observations faites par le robot. Pour la suite de ce paragraphe nous définissons plusieurs icônes :

- position supposée du but.
- position réelle du but.
- position supposée du robot.
- position réelle du robot.
- obstacle.
- cellule libre.

Nous allons également introduire un formalisme : lorsque nous donnerons le résultat d'une observation, nous donnerons 0 si libre et 1 s'il y a un obstacle et nous fournirons dans l'ordre suivant l'état de la zone à gauche (cf figure 4.4), de la zone avant gauche, de la zone frontale, de la zone avant droite et de la zone droite. Par exemple, une observation où nous avons des obstacles de tous les côtés sauf devant donnera : 11011.

La figure 5.5 représente la carte de l'environnement que nous avons utilisé.

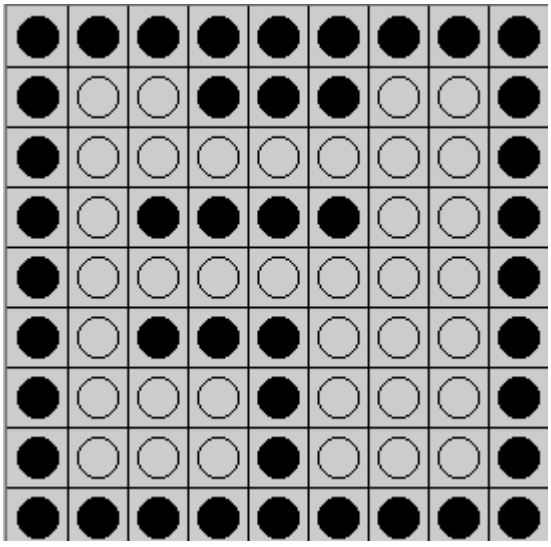


Figure 5.5 *– Carte de l'environnement*

Chacune des figures qui suivent est composée d'un cadre supérieur qui expose la planification calculée, dans le cadre intermédiaire : la distribution de probabilité en sachant que plus la teinte est foncée plus la probabilité est forte et inversement. Le cadre inférieur donne la position supposée du robot et la position supposée du but. Nous avons ajouté par un artifice graphique la position réelle du robot et du but.

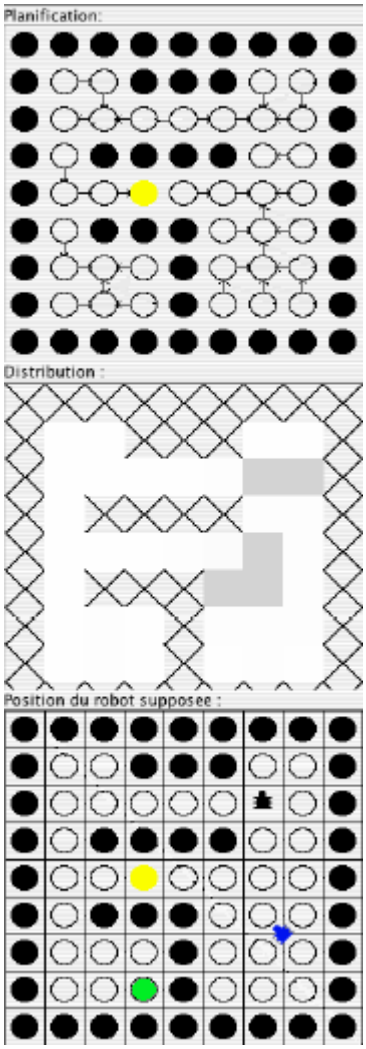


Figure 5.6 – *Première étape*

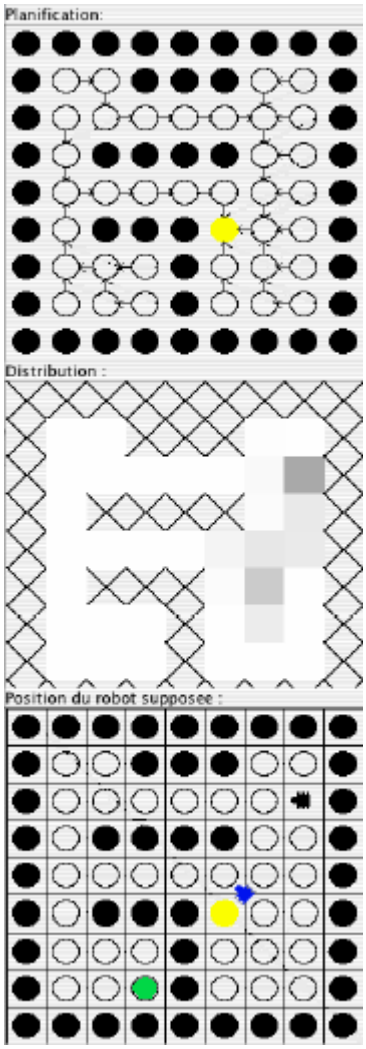


Figure 5.7 *– Première étape*

Cette première étape consiste en une simple observation qui donne 00000. Le senseur fournit (2, -3). Nous remarquons que l'environnement et la position initiale du robot sont identiques à la figure 4.15. Cependant, l'information issue du senseur permet de faire émerger des probabilités plus importantes.

Puis, nous effectuons le déplacement « Est-Est » et nous obtenons l'observation 00000. Le senseur fournit (3, -2). Nous remarquons que la localisation de la position supposée a été très rapide.

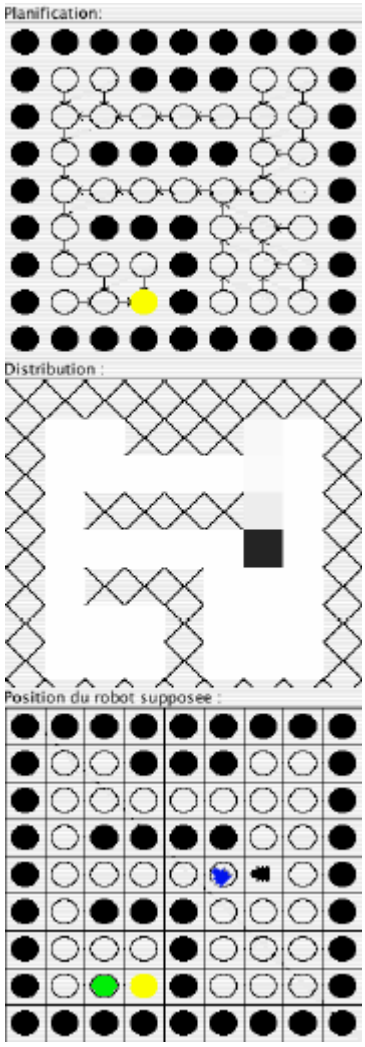


Figure 5.8 – *Troisième étape*

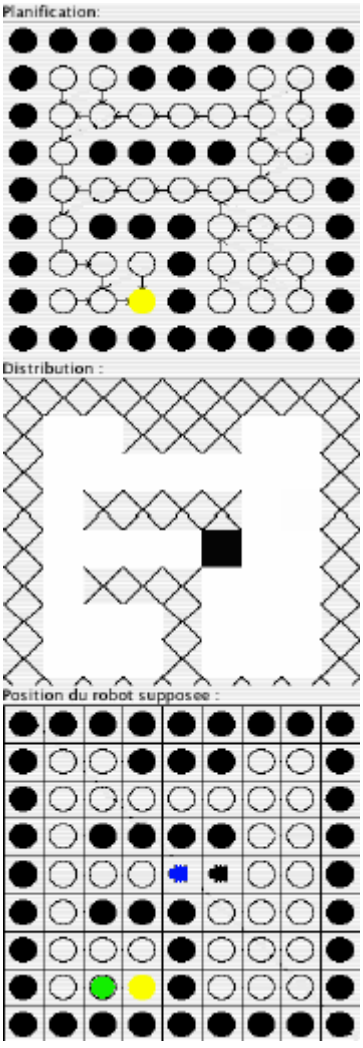


Figure 5.9 – *Quatrième étape*

Puis, nous effectuons le déplacement « Ouest-Ouest » et nous obtenons l'observation 00010. Le senseur fournit (3, -3). Nous remarquons que la probabilité est concentrée à son maximum grâce à la discrimination de l'observation. Nous connaissons la position du robot et donc celle de la cible.

Puis, nous effectuons le déplacement « Ouest-Ouest» et nous obtenons l'observation 11011. Nous remarquons que la probabilité demeure importante grâce à la forte discrimination de l'état où se trouve le robot

Nous remarquons dans l'ensemble des expérimentations que nous avons faites que le modèle et la détermination des paramètres sont bien adaptés à ce type de problèmes. Nous localisons plus rapidement le robot avec une marge d'erreurs faible grâce à l'information supplémentaire sur la distance relative au but. Lorsque ces erreurs surviennent : bruits sur les capteurs, incertitude de déplacements, … Nous retrouvons aisément la position du robot en quelques déplacements.

Nous pouvons remarquer que lorsque la cible bouge, la planification reste à peu près constante exceptée aux abords de la cible. Donc lorsque le robot se rapproche dans cette zone, les actions changent relativement souvent et perturbent le déroulement de l'exécution. Afin de remédier à ce petit problème, nous pourrions utiliser la caméra du Koala. Lorsque la cible devient visible, il abandonne l'exécution de la planification et poursuit la proie grâce au système vidéo.

5.4. Conclusion

L'approche présentée a le grand intérêt de permettre au robot d'agir avant que le plan optimal ne soit calculé. En utilisant cette technique, nous pouvons espérer atteindre le but plus rapidement. Grâce à cette technique, il est possible d'atteindre un but mobile facilement. En effet, à chaque pas de planification, nous recalculons le plan en fonction de la nouvelle position du but.

Nous remarquons que la méthode que nous avons exposée est relativement bien adaptée au cas de figure que nous avons envisagé : recherche et poursuite d'une cible mobile. Le comportement que nous avons observé est assez proche de celui constaté lorsqu'un prédateur chasse sa proie.

Cependant, la détermination des différents paramètres d'exécution a une grande importance. Elle nécessite la prise en considération de données techniques majeures. Leurs valeurs engendreront la réussite ou l'échec de l'expérience sans toutefois prouver la validation ou, respectivement, l'absurdité du modèle.

6. Conclusions et perspectives

Pour conclure ce rapport, nous commencerons par un bref résumé soulignant les apports, avant d'indiquer les différentes perspectives que nous pouvons envisager.

6.1. Résumé et apports

Nous avons présenté dans ce rapport une étude de l'utilisation de modèles stochastiques pour planifier les actions d'un robot mobile évoluant dans un environnement intérieur structuré. Si dans un premier temps nous nous sommes surtout intéressés au modèle le plus général (les Processus Décisionnels de Markov Partiellement Observés ou POMDP), qui permettent de prendre en compte toutes les incertitudes dès la phase de planification, nous avons rapidement restreint notre champ d'étude à un cas particulier de ces modèles : les Processus Décisionnels de Markov Parfaitement Observés (MDP).

Notre apport dans le cadre des MDP se situe à plusieurs niveaux :

- Dans le chapitre 3, nous montrons l'importance de l'adaptation des différents paramètres afin d'obtenir des politiques intéressantes dans le cadre de la robotique mobile. Nous montrons dans ce chapitre quels facteurs sont importants pour obtenir ce que nous appelons des politiques intéressantes pour la robotique : à la fois efficaces et sûres. Elles n'hésitent pas à préconiser des détours pour éviter des chemins obstrués qui mettraient en danger le robot, compromettant ainsi le succès des missions allouées. La fonction de transition doit refléter au mieux les comportements réels du robot, les obstacles doivent être suffisamment répulsifs et le facteur d'atténuation γ fixé avec soin. Enfin, dans ce même chapitre, nous comparons sur un petit exemple le fonctionnement des deux algorithmes généralement utilisés pour résoudre les MDP.

- Dans le chapitre 4, nous présentons le robot que nous avons utilisé pour les expérimentations. Nous détaillons la technique de localisation que nous avons appliquée à notre problème. Nous explicitons l'adaptation de la fonction d'observation pour l'exécution réelle. Ensuite, nous exposons le cadre d'un environnement réel et développons l'utilisation du robot en pratique. Nous présentons sa communication. Nous définissons un déplacement du robot. Nous explicitons les cas litigieux et leurs résolutions. Puis nous présentons l'évitement d'obstacles et l'abstraction des données fournies par les capteurs. Enfin, nous présenterons une expérimentation de cette exécution en réelle.

- Dans le chapitre 5, nous présentons nos travaux qui abordent une nouvelle approche. Il s'agit, pour le robot d'atteindre un but mobile dont la position est inconnue. La position initiale du robot n'est pas connue et la seule information qu'il dispose est la position relative par rapport au but. Comme la position initiale du robot est inconnue, en fonction des observations du robot, nous allons avoir émettre des hypothèses sur sa position. L'information qu'il dispose sur le but, va nous aider à discriminer ou conforter

ces hypothèses. Nous allons faire une itération de policy itération avec les informations dont nous disposons sur le but. Puis nous exécutons l'action correspondante et nous recommençons la stratégie depuis le début.

La thématique de recherche de ce stage de DEA s'est avérée très intéressante et le travail accompli a été également très enrichissant pour moi.

6.2. Perspectives

Nous avons présenté les principales perspectives que nous pouvons envisager pour la poursuite de ce travail.

6.2.1. Techniques d'approximation

Nous avons expliqué auparavant que l'application des Processus Décisionnels de Markov Partiellement Observés à des problèmes de taille réaliste se révélait impossible. C'est pourquoi les récentes études dans cette thématique de recherche sont fondées sur les Processus Décisionnels de Markov Parfaitement Observés. Mais les algorithmes classiques de résolution de MDP présentés sont complexes, ce qui les rend difficilement adaptables à des environnements de grande taille, tels que ceux nécessités dans le cadre de la robotique. En effet, lors de différents tests de notre application sur de grands environnements, la mémoire de l'ordinateur utilisé n'a pu suivre les besoins nécessaires aux algorithmes de calcul du plan. De ce fait, les différents travaux actuels s'intéressent aux techniques permettant d'obtenir plus rapidement des politiques sous-optimales mais néanmoins intéressantes. Nous pouvons ainsi citer les travaux de Pierre Laroche [Laroche, 2000] sur l'agrégation d'états et la décomposition des Processus Décisionnels de Markov ou encore les différentes approches exposées dans l'article de [Boutilier *et al.*,1999].

6.2.2. Améliorations amenées à la localisation

De nombreuses extensions se présentent à nous. Nous pouvons notamment développer par la suite au niveau de l'exécution, la possibilité de choisir parmi plusieurs stratégies de localisation du robot. En effet, notre stratégie est, pour le moment, uniquement fondée sur la stratégie de localisation de l'état le plus probable. L'article de Cassandra, Kuelbing et Kurien [Cassandra et al.,1996] détaillent plusieurs stratégies et comparent leurs différents résultats expérimentaux suivant la connaissance précise ou non de la position initiale du robot et des environnements plus ou moins différents. Trois stratégies ressortent des expérimentations par leurs meilleurs résultats : MLS (*most likely state*) notre stratégie actuelle d'état le plus probable, la méthode Q-MDP et Replan. Il serait donc intéressant au niveau de l'exécution d'implémenter ces deux autres stratégies.

6.2.3. Améliorations liées au robot

Par ailleurs, nous avons défini les observations du Koala pour cinq zones situées à l'avant du robot et sur ses côtés. Mais en réalité, le robot a également des capteurs à l'arrière, comme le montre la figure 4.2, dont nous n'en tenons pas compte dans notre étude. L'ajout de trois nouvelles zones d'observation situées à l'arrière du robot est également une priorité dans nos prochains développements, car à partir de ces observations supplémentaires la localisation du robot sera meilleure. Il faut pour cela tout d'abord modifier le nombre d'observations, puis élaborer une nouvelle fonction d'observation.

Mais, même avec ces modifications notre fonction d'observation restera assez simpliste et la localisation de notre robot sera très loin d'être parfaite. De ce fait, parmi les travaux qu'il reste à effectuer, nous pouvons citer l'apprentissage de la fonction d'observation.

6.2.4. Intégration d'un niveau symbolique

Dans ce rapport, nous nous sommes principalement intéressés à la planification d'actions, sans nous soucier véritablement des buts réels du robot. Or si le robot se déplace d'un point à un autre, c'est pour remplir une mission, par exemple le dépôt du courrier. Pour traiter ce type de mission, il serait nécessaire, d'intégrer notre module de planification d'actions à un planificateur de tâches plus classique, mais prenant tout de même en compte l'aspect probabiliste des actions.

6.2.5. La cible

Dans cette étude, nous utilisons un senseur afin d'obtenir la position relative de la cible mobile par rapport au robot. Or, nous avons fourni au robot des informations précises puisque nous ne disposions pas de ce capteur. Donc nous pouvons envisager, par la suite, l'utilisation d'un capteur réel. Alors, nous devrons tenir compte des données imparfaites qu'il nous fournira. De plus, lorsque la cible devient visible pour le robot, nous pourrions coupler notre méthode à une poursuite visuelle de la proie.

Nous nous sommes limités à la considération d'une seule cible mobile. Mais, nous pourrions également opter pour la présence de plusieurs buts mobiles et même considérer l'existence de cellules répulsives. Ainsi, nous nous positionnerons non plus du côté du prédateur mais de celui de la proie.

Bibliographie

[Bellman, 1957] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[Boutilier *et al., 1999*] Craig Boutilier, Thomas Dean et Steeve Hanks. Decision-theoric planning : Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11 :1-94, 1999.

[Cassandra et al.,1996] Anthony R. Cassandra, Leslie P. Kaelbling et James A. Kurien. Acting under uncertainty : Discrete bayesian models for mobile-robot navigation *Proceedings of IEEE International Conference on Intelligent Robots and Systems*,1996

[Dean et al.,1993] Thomas Dean, Leslie Kaelbling, Jak Kirman et Ann Nicholson. Planning with deadlines in stochastic domains. Dans *Proceedings of the 11th National Conference of Artificial Intelligence*, 1993.

[Dieter et al.,1999] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 1999.

[Howard, 1960] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachussets, 1960.

[Laroche, 2000] Pierre Laroche. *Processus Décisionnels de Markov appliqués à la planification sous incertitude.* Mémoire de thèse, Université Henri Poincaré Nancy 1, 2000.

[Littman et al.,1995b] Michael L.Littman, Thomas L.Dean et Leslie Kaelbling. On the complexity of solving markov decision problems. *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95), Montreal, Québec, Canada*, 1995.

[Littman, 1996] Michael L.Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, 1996.

[Puterman, 1994] Martin L.Puterman. *Markov Decision Processes*. John Wiley & Sons, New York, 1994.

[Tijms,1986] Henk C.Tijms. *Stochastic Modeling and Analysis, A Computational Approach.* John Wiley & Sons, New York, 1986.

[Watkins et Dayan, 1992] C.J.C.H Watkins et P.Dayan. *Q-Learning. Machine Learning*, 8(3) :279-292,1992.

[TAPUS, 2002] Adriana TAPUS. *Utilisation de processus de décision markoviens pour la planification et l'exécution d'actions par un robot mobile*, DEA ISC, 2002.