INRIA
RHÔNE-ALPES

Université Joseph Fourier

U.F.R Informatique &
Mathématiques Appliquées

I. M. A. G.
DEA D'INFORMATIQUE :
SYSTEMES ET COMMUNICATIONS

Projet présenté par :

Adriana TAPUS

Utilisation de processus de décision markoviens pour la planification et l'exécution d'actions par un robot mobile

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

Date : 20 juin 2002

Jury : N. HALBWACHS
F.RECHENMANN
C.LABBE
O.AYCARD

Sommaire

Résumé

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

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 obtenus par des méthodes déterministes.

C'est dans ce cadre que s'inscrit ce stage. Nous nous sommes intéressés ainsi à une étude théorique des Processus Décisionnels de Markov et à une application de ceux-ci 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 de position inconnue, par le robot. Toutes les expérimentations ont été validées par le prototype que nous avons mis en place. Le prototype est disponible à l'adresse URL : http://www-leibniz.imag.fr/LAPLACE/ENGLISH/MEMBERS/MEMBERSPAGES/tapus.html.

Par ailleurs, de nombreuses perspectives se présentent, nous pouvons citer par exemple le test en réel avec le robot, la possibilité de choisir la stratégie de localisation lors d'une exécution, la prise en compte des capteurs situés à l'arrière du robot, l'amélioration de la navigation du robot pour le moment pas très réaliste ou encore de nombreuses extensions au niveau graphique du point de vue pédagogique.

Les premiers résultats expérimentaux montrent clairement que l'application à des environnements assez grands n'est pas possible et que les algorithmes de planification utilisés restent complexes. C'est pourquoi, les dernières études se penchant sur le domaine, s'intéressent aux techniques permettant d'obtenir plus rapidement des plans sous-optimaux mais intéressants.

Abstract

Keywords: Planning, Markov Decision Processes, Markov Localization, Robotics.

A mobile robot needs to take into account various sources of uncertainty in order to perform robustly in a real environment. Its actions are not always reliable, and the robot might have problems with observing things around it, due to noise generated by its sensors. Stochastic models such as Markov Decision Processes (MDPs) are well suited to cope with uncertainty, and they are studied for some time now in the community of Artificial Intelligence.

So, during this paper, we studied the MDPs and we focused on the application of the MDPs in robotics. We have been able to show the limits and the advantages of these models. We have also added a new approach, in which the robot searches the goal that is in an unknown position. The only knowledge that we have are the observations of the robot and the relative distance of this one to the goal. All the examples shown in this paper were extracted from our prototype.

Some of the future directions of this work include: tests on the real robot, the possibility to change the localization algorithm on the fly, the improvement of the robot navigation system or some extensions of the human computer interaction interface.

In conclusion, the first experimental results show us clearly that the application for large environments, with many states, is impossible, and that the planning algorithms that we have used, are intractable. As a result, much research focuses on approximation techniques trying to reduce the search space to obtain near-optimal policies in a reduced computing time.

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 donné la chance 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 et en particulier son responsable Pierre Bessière.

Mes remerciements vont aussi aux différents membres de mon jury :

- Monsieur Nicolas Halbwachs, Directeur de recherche CNRS
- Monsieur François Rechenmann, Directeur de recherche INRIA
- Monsieur Cyril Labbé, Maître de Conférences à l'Université Joseph Fourier de Grenoble

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, si les robots de première génération qui étaient des automates, étaient uniquement destinés à répéter une suite de mouvements au sein d'un environnement statique, n'avait 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 comme par exemple le glissement de ses roues, la rencontre d'obstacles mobiles, imprévus ou encore l'incertitude due 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'I.A. sont apparues, concernant l'utilisation d'une certaine classe de modèles stochastiques très adaptés à la prise en compte d'incertitudes : les Processus Décisionnels de Markov (PDM). Ceux-ci 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 brièvement les domaines auxquels nous nous sommes intéressés : la planification et les modèles stochastiques. Le but n'est pas de faire un historique des travaux de planification, mais simplement de montrer que de nombreux problèmes d'incertitudes se posent, motivant ainsi l'utilisation des modèles stochastiques, que nous présenterons par la suite.
- 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 celui-ci 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 l'exécution de politiques en présentant les différents moyens mis en place pour traiter une exécution. Nous introduirons également les modules qui nous permettront de gérer la localisation du robot.
- Dans le chapitre 5, nous introduirons une nouvelle approche : le robot part d'une position inconnue et doit atteindre un but dont la position est elle aussi inconnue. Les seules informations que nous détenons sont les observations faites par le robot et la distance relative de celui-ci 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 la planification et 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 rappellerons brièvement ce qu'est la planification et quels sont les problèmes qu'elle pose. Puis nous exposerons les modèles stochastiques auxquels nous nous sommes plus 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 que nous décrirons brièvement à la fin de ce chapitre.

2.2. Planification

La planification est une discipline à part entière de l'*Intelligence Artificielle*, et cela depuis une quarantaine d'années. Ce domaine s'intéresse à l'élaboration de systèmes capables de générer de manière automatique des suites d'actions. Celles-ci (appelées plans) ont pour but de faire passer l'univers de son état initial à un état final satisfaisant le but fixé au préalable. Face à ce type de problèmes, les questions suivantes se posent :

- De quelles connaissances dispose-t-on dans le domaine d'application ?
- Quelles sont les actions autorisées et comment les modéliser ?
- Quels sont les buts à satisfaire ? Y-a-t'il des buts prioritaires ?
- Quels algorithmes utiliser pour générer ces plans rapidement ?

Cette liste n'est bien entendue pas exhaustive. Les connaissances que nous avons sur le domaine sont très importantes : les ignorer pourrait mener à la génération de plans totalement inutiles. Selon les domaines d'application, ces connaissances peuvent être évidentes et faciles à réunir (par exemple, un robot est jusqu'à preuve du contraire incapable de traverser un mur !), ou au contraire nécessiter une phase longue et difficile d'acquisition auprès d'un expert du domaine.

Dès le début des recherches faites en planification, les études se sont naturellement tournées vers la robotique. Les premières applications permettaient la résolution de problèmes simples, tels que l'empilement de blocs sur une table, le célèbre monde des blocs. Les études se sont par la suite orientées vers la planification des déplacements des robots, également nommé *planification de trajectoires* qui consiste à générer un plan permettant à un robot,

placé dans un environnement pouvant être inconnu, de rejoindre un but en évitant les obstacles.

Ces divers travaux de planification dans le domaine de la robotique ont rapidement mis en lumière un des problèmes de l'approche déterministe de la planification : la non-prise en compte des incertitudes. En effet, les plans sont élaborés en partant de l'hypothèse que toute action est déterministe ce qui est rarement le cas dans un environnement réaliste et dynamique où l'exécution d'un plan est soumise à de nombreuses incertitudes :

- Incertitude dans les effets des actions. Suivant la structure du sol, les roues d'un robot peuvent patiner et de ce fait parcourir une distance moins grande que prévue. La présence d'obstacles inconnus lors de la génération du plan peut également fortement perturber la bonne exécution du plan.
- Incertitude sur la position initiale. Il est difficile de toujours connaître précisément la position et l'orientation du robot. Si une petite erreur n'est pas grave à court terme, elle peut en revanche avoir des conséquences fâcheuses après l'exécution d'une longue séquence d'actions.
- Incertitude des données des capteurs. Dans une application réelle de robotique mobile, le robot perçoit son environnement grâce à ses capteurs. Ceux-ci sont généralement plus ou moins bruités, et nécessitent une interprétation.

L'ensemble de ces incertitudes fait que l'exécution d'un plan ne se déroulera jamais comme prévue.

La nécessité de prendre en compte ces incertitudes a donné lieu à un nouveau type de planification (appelée *planification probabiliste*).

2.3. 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, on définit 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 :

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

Ce qui peut être explicité de la façon suivante : l'état du système à l'instant t dépend uniquement de l'état à l'instant $t - 1$. On peut généraliser cela 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 :

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

2.4. Les Processus Décisionnels de Markov

2.4.1. Cadre théorique

Les incertitudes vont être gérées en utilisant le Processus Décisionnel de Markov Parfaitement Observé (que nous désignerons par la suite par le terme de PDM) [Bellman,1957], qui 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 : $T : S \times A \times S \_ [0,1]$

On note $T(s,a,s')$ 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 celle-ci sera supprimée lors de l'exécution d'un plan, comme nous le verrons au chapitre 4. Cette simplification a le grand avantage de permettre le calcul de plans sur un espace d'états fini, ce qui réduit considérablement la complexité des algorithmes. Pour ce type de modèle, nous parlerons aussi de politique pour définir un plan.

2.4.2. Politiques dans le cadre des Processus Décisionnels de Markov

Critère d'optimalité

La politique optimale est calculée en fonction de la fonction de gain : il s'agit d'optimiser les récompenses possibles. Nous distinguons deux critères : celui de la récompense moyenne (*average-reward criterion*) et celui de la récompense totale pondérée (*discounted infinite horizon reward*). Ce dernier critère est le plus utilisé et il est très intéressant car 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. C'est précisément le coefficient _ introduit dans le critère d'optimalité qui permet de faire le compromis voulu. Ce paramètre est un réel compris entre 0 et 1, pour nous donner la convergence de l'algorithme. Aussi, si nous faisons varier ce paramètre, nous obtenons différents types de politiques, chacune ayant son propre comportement. Dans la suite de ce rapport, c'est donc le critère d'optimalité de récompense totale pondérée que nous utiliserons.

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 ($\pi : S \_ A$). Calculée sur un horizon infini, la politique optimale est stationnaire, c'est à dire qu'elle est indépendante du temps.

2.4.3. Algorithmes de calcul de politiques

Il existe plusieurs algorithmes permettant de calculer une politique optimale dans le cadre des PDM. 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 on ajoute les valeurs des différents états qu'il est possible d'atteindre en exécutant cette même action tout en prenant en compte le facteur _ 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 :

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

Comme il permet un calcul itératif, nous pouvons dire alors qu'il est un algorithme *anytime*[1].

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

Algorithme de *Value Iteration* :

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

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

[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.

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 _. La fonction de valeur obtenue est alors sous-optimale mais plus _ 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|, _ 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 3.2.5**.**

L'algorithme Policy Iteration

Ce deuxième algorithme différent du précédent 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* calculant 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 :

$$V_\pi(s) = R(s) + \_ \sum_{s' \_ S} T(s, \pi(s), s') \text{ x } V_\pi(s') \qquad (2.3)$$

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 un peu comme dans *Value Iteration* pour chaque état une action améliorant la valeur courante. 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 $s \_ S$ **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 $s \_ S$ **faire**
si il existe une action $a \_ A$ telle que :
$R(s) + \_\ \Sigma_{s' \_ S}\, T(s,a,s') \times V_\pi(s') > V_\pi(s)$
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 fonction de valeur. En pratique, cet algorithme nécessite un grand nombre d'itérations pour converger, mais chaque itération est très rapide.

- *Policy Iteration* améliore beaucoup la fonction de valeur 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 3.2.5.

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.

2.4.4. Application à la robotique

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 présentés ci-dessus, 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, dans les chapitres suivants.

Les PDMPO ne raisonnent plus sur les états mais sur les distributions de probabilités de ces états, c'est-à-dire qu'à chaque distribution correspond une action. L'application à des

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

problèmes de taille réaliste se révèle impossible avec les PDMPO. C'est pourquoi nous avons choisi les PDM (cas particulier de PDMPO), qui comme nous avons vu précédemment permettent de réaliser des algorithmes avec une complexité polynomiale, et donc qui les rendent tout à fait utilisables en pratique.

2.5. 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, trouvant ainsi un compromis entre l'efficacité et la sécurité, ce qui est particulièrement intéressant en robotique.

Aussi, ce cadre théorique met en œuvre suffisamment de paramètres pour que nous puissions l'adapter à notre application, afin de pouvoir 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, et cet intérêt n'a cessé depuis, 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 et dont ce stage s'est principalement inspiré.

Un Processus Décisionnel de Markov est un modèle qui peut être paramétré de diverses manières et il y a des choix à faire 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 mais un PDM doit également 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 _.

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, afin d'atteindre une position précise de cet environnement. 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 exemple de petit 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.

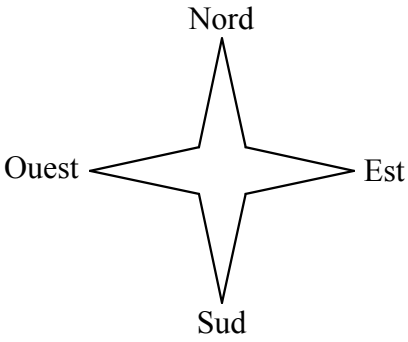


Figure 3.0 *– Boussole*

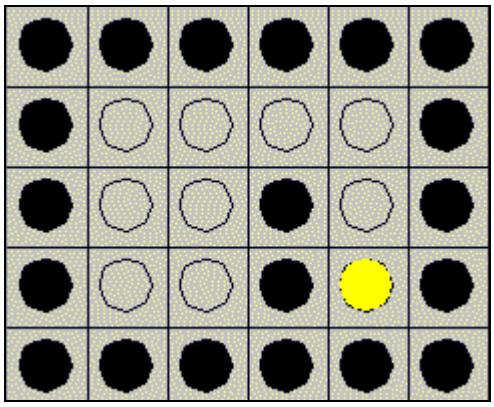


Figure 3.1 *– Petit environnement (9 états libres)*

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 du robot Koala, qui est le robot auquel nous nous sommes intéressés (voir Annexe1).

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 des 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 dans lequel se trouve le robot est proche du but, plus la récompense doit être forte. Cela 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, 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 l'on s'éloigne du but. Ainsi, il apparaît qu'il suffit 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. Ceux-ci 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 et ceci 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 avec une valeur identique pour chaque obstacle et chaque but, bornant strictement toutes deux 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, c'est grâce à cette fonction que nous prenons en compte l'incertitude quant aux effets 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 est générée expérimentalement.

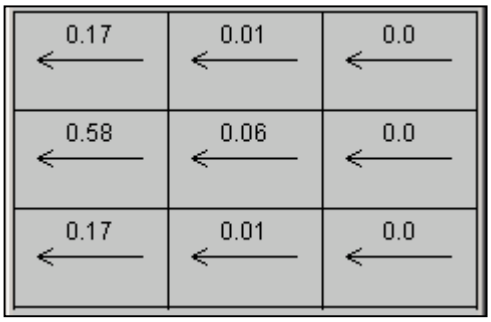


Figure 3.2 *– Fonction de transition pour l'action avancer ouest*

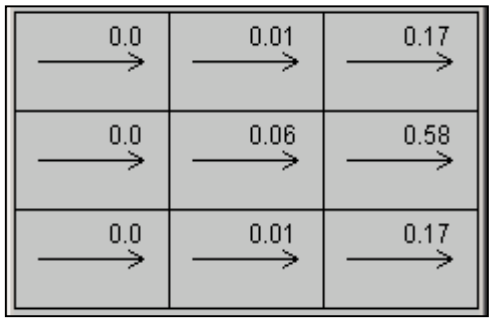


Figure 3.3 *– Fonction de transition pour l'action avancer est*

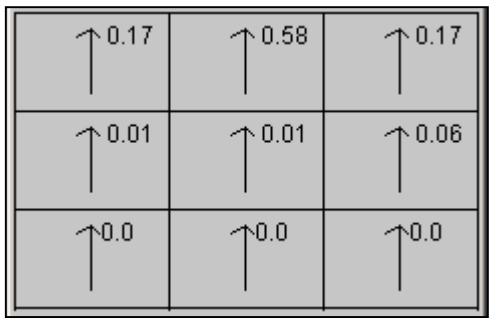


Figure 3.4 *– Fonction de transition pour l'action avancer nord*

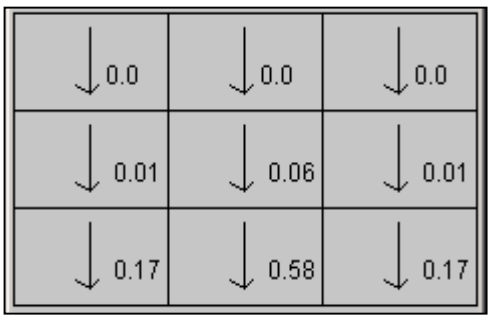


Figure 3.5 *– Fonction de transition pour l'action avancer sud*

Pour la fonction de transition que nous avons illustrée dans les figures 3.2, 3.3, 3.4, 3.5, nous avons choisi neuf cases, qui 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 la fonction de transition pour l'action avancer ouest, 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 nous pouvons l'apercevoir dans la figure 3.6, et cette représentation des neuf cases 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, quelle que soit sa direction initiale, 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 _

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 2.4.2). En effet, si on ne s'intéresse qu'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, celui qui cherche à maximiser l'espérance pondérée de gain futur (*discounted infinite horizon reward*) est le plus souvent utilisé. Celui-ci donne plus d'importance 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 suivi lui fait faire un détour de 100 mètres également, c'est beaucoup plus fâcheux. Ici dans un 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, c'est donc celui-ci que nous allons utiliser par la suite.

Le paramètre _ 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 cela 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 _ différentes : 0.70 à gauche et 0.95 à droite. L'action avancer est représentée par une flèche reliant un cercle à un autre et orientée vers le nord si nous effectuons 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.

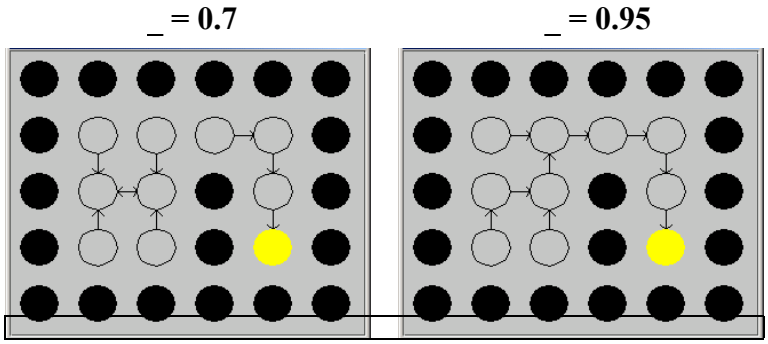


Figure 3.7 *– Politiques obtenues pour différentes valeurs de _*

Ces 2 politiques diffèrent pour les trois positions en haut : (1,1), (1,2) et (2,2). En effet sur la figure à gauche nous pouvons voir que le robot préfère par exemple dans l'état (2,2) d'avancer à l'ouest 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 _ assez proche de 1. Ainsi dans tous les exemples qui suivent nous avons fixé la valeur de _ à 0.99, car celle-ci permet d'obtenir des plans satisfaisants dans le type d'environnement que nous utilisons. Il faut également noter que plus _ est proche de 1, plus le nombre d'itérations nécessaires à l'obtention du plan est grand.

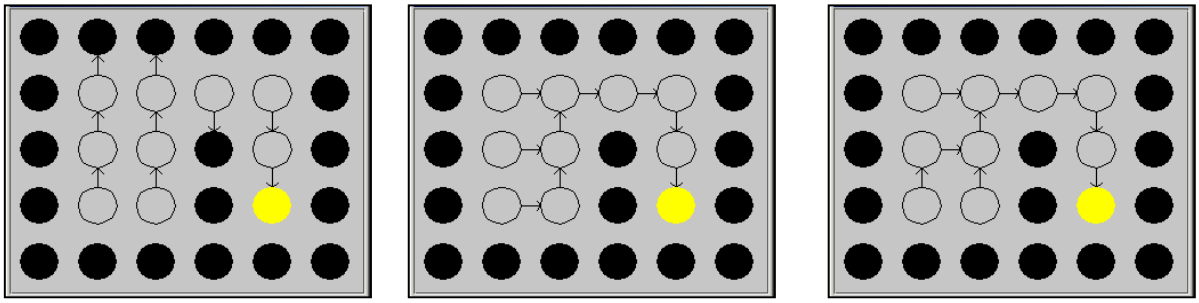
Enfin, notre prototype étant un simulateur, cette valeur doit être facilement modulable et ceci pour chaque environnement afin de pouvoir comparer les différentes politiques obtenues.

3.2.5. Choix de l'algorithme de résolution

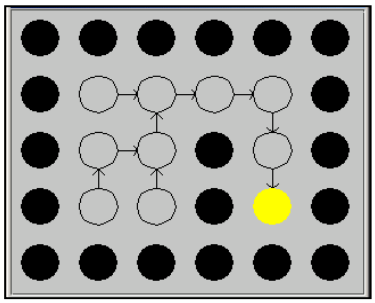
Les deux algorithmes classiques de résolution de Processus Décisionnel de Markov ont chacun leurs supporters : selon Puterman [Puterman, 1994] , Value Iteration est plus rapide, mais Tijms [Tijms, 1986] favorise Policy Iteration. En fait, les deux algorithmes ont des fonctionnements très différents : 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'y a 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 2.4.3, 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.



Politique initiale *Etape 1* *Etape 2*



Etape 3

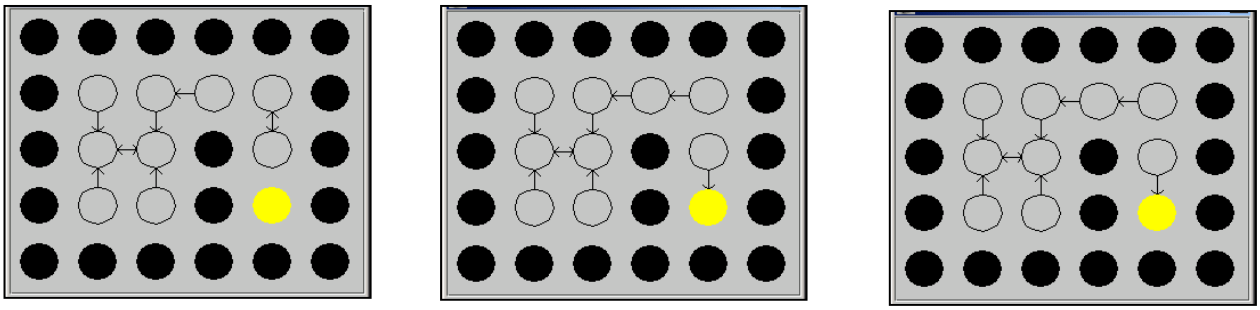
Figure 3.8 *– Policy Iteration : d'une politique aléatoire à la politique optimale*

Cette figure ne montre qu'un exemple de déroulement de l'algorithme. En effet, celui-ci 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 2.4.3). En effet, ici 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.

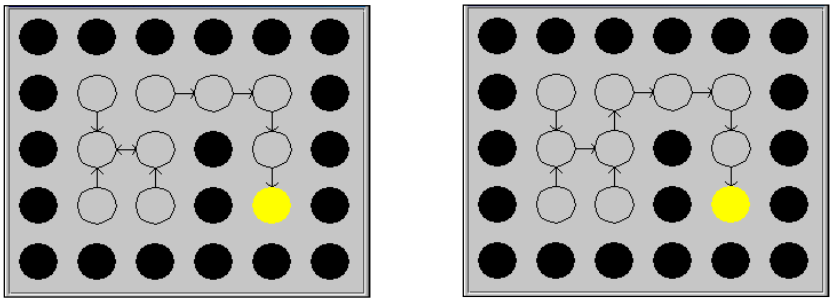
La figure 3.9 montre les trois premières itérations de l'algorithme.



Etape 1 *Etape 2* *Etape 3*

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

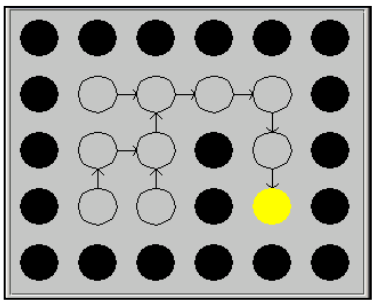
Nous pouvons ainsi remarquer dans la figure 3.9 que les trois politiques sont quasiment identiques. La figure 3.10 montre les premiers progrès de la politique qui n'interviennent qu'à l'itération 15 de l'algorithme. A chaque itération un choix est effectué entre les quatre actions de translation, qui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



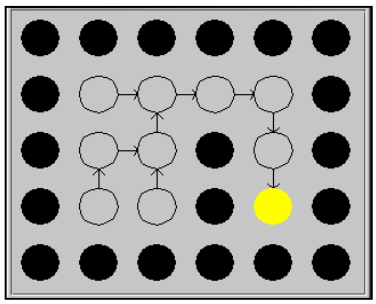
Etape 15 *Etape 16*

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 des petites améliorations successives de la fonction de valeur.

Conclusion

Ce petit exemple ne constitue en aucun cas une preuve de la supériorité de Policy Iteration vis-à-vis de Value Iteration. Par exemple, sur le même problème, en faisant uniquement passer le facteur _ de 0.99 à 0.9, l'écart entre les deux algorithmes a fortement diminué : le nombre d'itérations de Value Iteration a diminué, pour un temps de calcul de 1,7 secondes, alors que les temps de calcul de Policy Iteration n'ont pas varié. Ceci montre bien l'importance des paramètres du PDM sur l'efficacité de ces algorithmes.

3.2.6. Autres points de programmation concernant les plans.

Dans la suite de sa thèse, 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%.

Aussi, pour notre prototype, nous avons donné à l'utilisateur le choix d'initialiser l'algorithme *Policy Iteration* soit par une politique aléatoire, soit par la politique du plus court chemin de l'environnement support. De plus, étant donné l'efficacité de la politique du plus court chemin et son rapide calcul, la possibilité de générer des plans directement à partir de cet algorithme a été ajouté à notre application.

Néanmoins, lorsqu'un des deux algorithmes d'itération est lancé, le temps de calcul de la politique peut être très long, plusieurs minutes. Face à ce problème, nous avons spécifié deux modes d'affichage parmi lesquels l'utilisateur fera son choix suivant ses attentes. Le premier donne à l'utilisateur la possibilité de suivre pas à pas l'évolution de la politique au cours des itérations, chaque nouvelle itération étant effectuée à la demande de l'utilisateur. Cette option, de nature pédagogique, permet de mieux comprendre l'algorithme utilisé et son fonctionnement.

Cependant, mis à part l'intérêt éducatif de cette option, son utilisation reste lourde du point de vue de l'utilisateur qui se trouve obligé de confirmer la poursuite de l'algorithme itératif jusqu'à l'obtention du plan final. Le second mode d'affichage répond à ce problème en offrant la possibilité à l'utilisateur de lancer l'algorithme de telle façon que le plan, ne soit affiché qu'une fois calculé.

Enfin, une troisième option d'affichage faisant le compromis entre ces deux dernières, aurait été intéressante. C'est à dire tel qu'il lancerait l'algorithme et afficherait à chaque itération la nouvelle politique obtenue en poursuivant le calcul. Avec cette possibilité, l'utilisateur peut donc s'absenter lors d'un calcul de plan, et pourra toujours observer l'évolution de la politique courante et plus ou moins savoir quand l'algorithme touche à sa fin lorsque la politique affichée est presque optimale. Cependant, nous ne sommes pas arrivés à mettre en place cette option car l'actualisation de l'affichage à chaque pas ne se fait pas correctement. Ce point reste donc une extension de plus à développer.

3.2.7. Comparaison entre deux PDM paramétrés différemment

Nous allons essayer de comparer deux modèles PDM, paramétrés différemment, et donc de justifier le choix de nos paramètres.

Le premier modèle PDM reste celui que nous venons de décrire en détail jusqu'ici. Comme deuxième modèle, nous choisissons le même que le premier, dans lequel nous modifions la fonction de transition et les actions, et nous introduisons les quatre directions cardinales.

Dans la suite, nous allons décrire un peu plus en détail le deuxième modèle. Chaque case de l'environnement va correspondre à quatre états définis selon les quatre directions cardinales. Nous allons contrôler le robot à travers trois actions atomiques: Avancer d'une case, Pivoter de +90 degrés et Pivoter de -90 degrés. La nouvelle fonction de transition va correspondre à la Figure 3.12 et à la Figure 3.13.

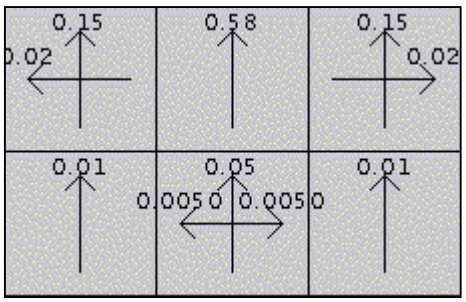


Figure 3.12 *– Fonction de transition pour l'action avancer*

Pour mieux comprendre cette nouvelle fonction, nous allons la décrire en quelques lignes. Nous commencerons par la fonction de transition pour l'action avancer. L'état du robot correspond à l'état au centre en bas, dirigé vers le nord (voir figure 3.12). Alors la probabilité que le robot se trouve dans la même case, sachant qu'il a effectué l'action avancer, orienté vers le nord est de 0.05, orienté vers l'est ou vers l'ouest est égale à 0.005. La probabilité que le robot se trouve une case au-dessus, après l'exécution de l'action avancer, orienté vers le nord, est égale à 0.58, et ainsi suite pour les autres cas.

Figure 3.13 *– Fonction de transition pour les actions pivoter gauche et pivoter droite*

Pour expliquer la fonction pour le pivotement, nous allons présenter celle qui correspond à l'action Pivoter à Gauche de 90°. Celle-ci peut être décrite de la manière suivante : la probabilité que le robot orienté initialement vers le nord, en voulant pivoter vers la gauche de 90°, de se trouver finalement encore orienté vers le nord est de 0.1, qu'il se trouve orienté vers la gauche est de 0.85 et qu'il se trouve orienté vers le sud est de 0.05. Nous expliquons de la même façon la fonction pour le pivotement à droite de 90°, avec la remarque que nous avons effectué une rotation vers la droite.

Ainsi au début de mon projet, je suis partie de ce deuxième modèle de PDM, paramétré de cette manière-ci. J'ai implémenté les deux algorithmes de planification : Value Iteration et Policy Iteration et j'ai rencontré deux problèmes qui m'ont déterminée à changer les paramètres mentionnés précédemment. Le premier problème est le nombre d'états de l'environnement : celui-ci étant égal au nombre de cases de l'environnement multiplié par

quatre (les quatre points cardinaux), nous arrivons alors, pour de grands environnements, à avoir un nombre d'états assez important, ce qui rend les calculs très lourds et les algorithmes difficilement adaptables. Et le deuxième problème est lorsque le robot se trouve dans une zone dangereuse (c'est à dire où il peut facilement rentrer dans un obstacle), il préfère pivoter autour de lui-même, au lieu d'avancer.

Nous pouvons voir ce comportement sur l'exemple suivant (voir figure 3.14) :

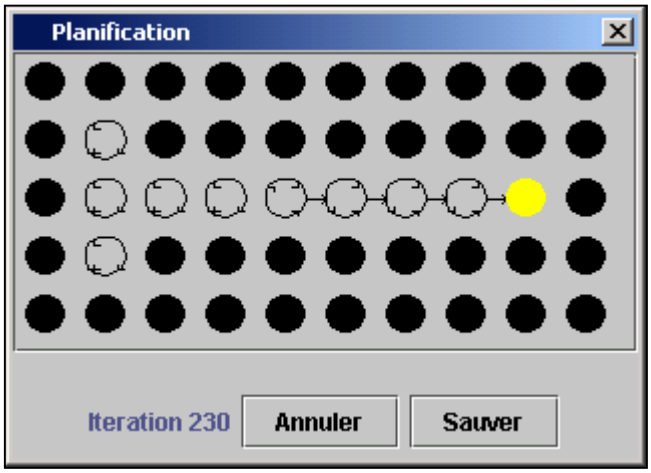


Figure 3.14 *– Calcul d'un plan avec le deuxième modèle PDM*

Comme nous pouvons le constater à travers l'exemple simple (figures 3.14 et 3.15), l'amélioration est conséquente. En effet, nous passons de 230 itérations à 17 itérations, et de plus le robot cesse de tourner sur lui-même lorsqu'il est dans une zone dangereuse, il trouve son chemin immédiatement et rejoint le but sans aucun problème. Un autre avantage par rapport au modèle abandonné est le fait que la planification soit indépendante de l'orientation du robot.

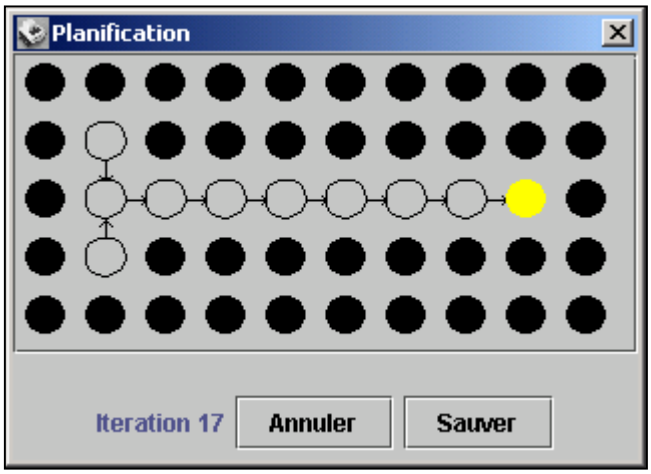


Figure 3.15 *– Calcul d'un plan avec le premier modèle PDM*

Le choix d'abandonner les anciens paramètres au profit de ceux présentés jusqu'ici, est complètement justifié, puisque les améliorations sont conséquentes et non négligeables.

3.3. Conclusion

Nous avons présenté dans cette partie l'adaptation des Processus Décisionnels de Markov à la robotique et nous avons étudié le modèle et les différents algorithmes.

Un des avantages des PDM que nous avons vus pendant ce chapitre 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 soit connue ou inconnue, elle est réalisée tout le temps de la même manière. Dans ce chapitre, 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, avantage que nous allons exploiter dans le chapitre 5. 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.

4. Exécution d'une politique

4.1. Introduction

Comme nous l'avons vu lors du chapitre précédent, 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. 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, ce qui rend la localisation très incertaine. De plus, 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, celui-ci 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], qui 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, dans ce chapitre 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. Dans une deuxième partie, nous détaillons comment nous avons appliqué cette technique à notre problème et illustrons cette application par des expérimentations. Enfin, nous conclurons. Ajoutons aussi que comme dans le chapitre précédent, l'algorithme a été implémenté dans notre prototype et que les expérimentations présentées sont issues de celui-ci.

---

[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.

4.2. Localisation markovienne

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 : $O : S \times \vartheta \rightarrow [0,1]$, où O(s,o) signifiant la probabilité d'observer o dans l'état s.

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 de nouveau 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 :

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

$\Pr_{t\text{-}1}(s)$ 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.

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.

4.3. Observations du robot

Lors d'une exécution en réel, le Koala observe son environnement à l'aide de seize capteurs infrarouges sensibles à la proximité et à la luminosité. Étant donné la répartition des capteurs sur le robot Koala, nous avons divisé cette « ceinture » de capteurs en cinq zones d'observation (cf. figure 4.1) : une zone frontale composée de six capteurs, une zone avant gauche et une zone avant droite comportant chacune un capteur et une zone de chaque côté du robot ayant chacune deux capteurs. 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.

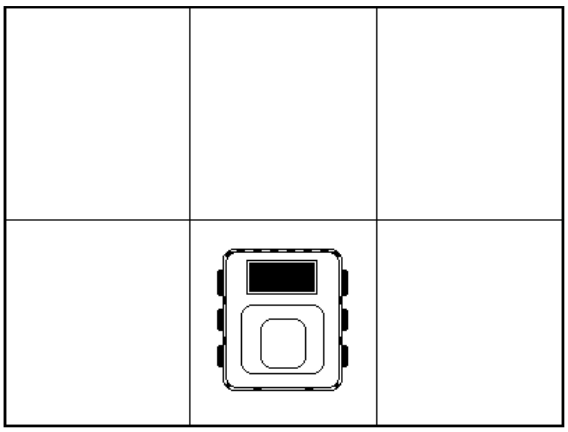


Figure 4.1 *– Zones physiques d'observation utilisées*

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.

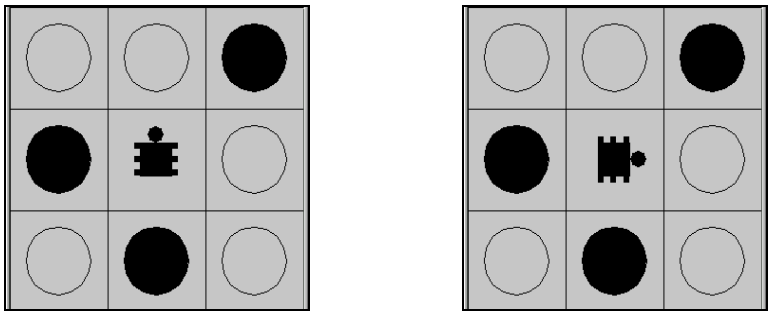
4.4. Fonction d'observation

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 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 utiliser par la suite une fonction d'observation qui va 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 :

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

Nous n'allons pas prendre en compte les orientations dans les probabilités des positions probables précédentes, car notre fonction de transition est indépendante de l'orientation.

Ainsi, si pendant la planification nous avons pu éliminer l'orientation du robot, ici pour l'exécution c'est inévitable. Les observations du robot dépendent de son orientation. Prenons l'exemple illustré dans la figure 4.2.



a *b*

Figure 4.2 *– 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 4.2a). Mais par contre si l'orientation du robot était Est, conforme la figure 4.2 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). Donc, 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 l'observation 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. Exécution en réel

Comme nous l'avons vu précédemment, les observations faites par le robot aident le robot 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, aussi, d'éviter des collisions pouvant endommager le robot.

Pendant l'exécution du plan, et afin d'éviter toute collision, le robot devra observer son environnement avant chaque action.

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.

Même quand le robot se perd, il arrive relativement souvent qu'après avoir exécuté plusieurs actions, il rencontre une observation discriminante dans l'environnement, ce qui lui permet de se relocaliser.

Pour la « communication » entre notre simulateur et le robot, dont les commandes sont implémentées en C++, nous avons opté pour une communication à travers des fichiers, ainsi notre simulateur détermine à chaque pas l'action à effectuer et la transmet au robot à travers un fichier. De son côté, le robot observe son environnement toutes les secondes et génère alors un fichier correspondant aux valeurs rendues par les capteurs, puis à partir de ce fichier nous calculons l'observation du robot.

4.6. Résultats

Comme nous l'avons mentionné tout à l'heure, nous avons intégré dans notre application deux types d'exécution qui diffèrent sur le fait de connaître ou non la position initiale du robot.

Nous allons décrire quelques tests, en tenant compte de ces deux types d'exécution.

Position initiale connue

Quand la position initiale du robot est connue, la probabilité qu'il soit dans l'état initial au moment t=0 est égale à 1, et la probabilité que le robot se trouve dans n'importe quel autre état au même moment est égale à 0.

Soit le plan de la figure 4.3 :

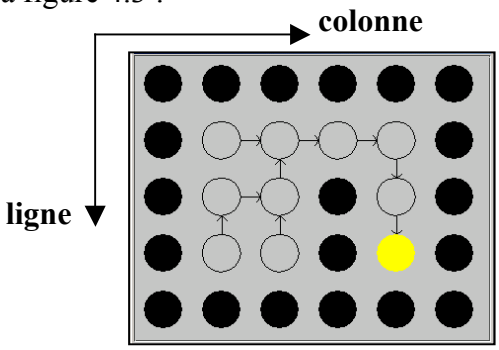


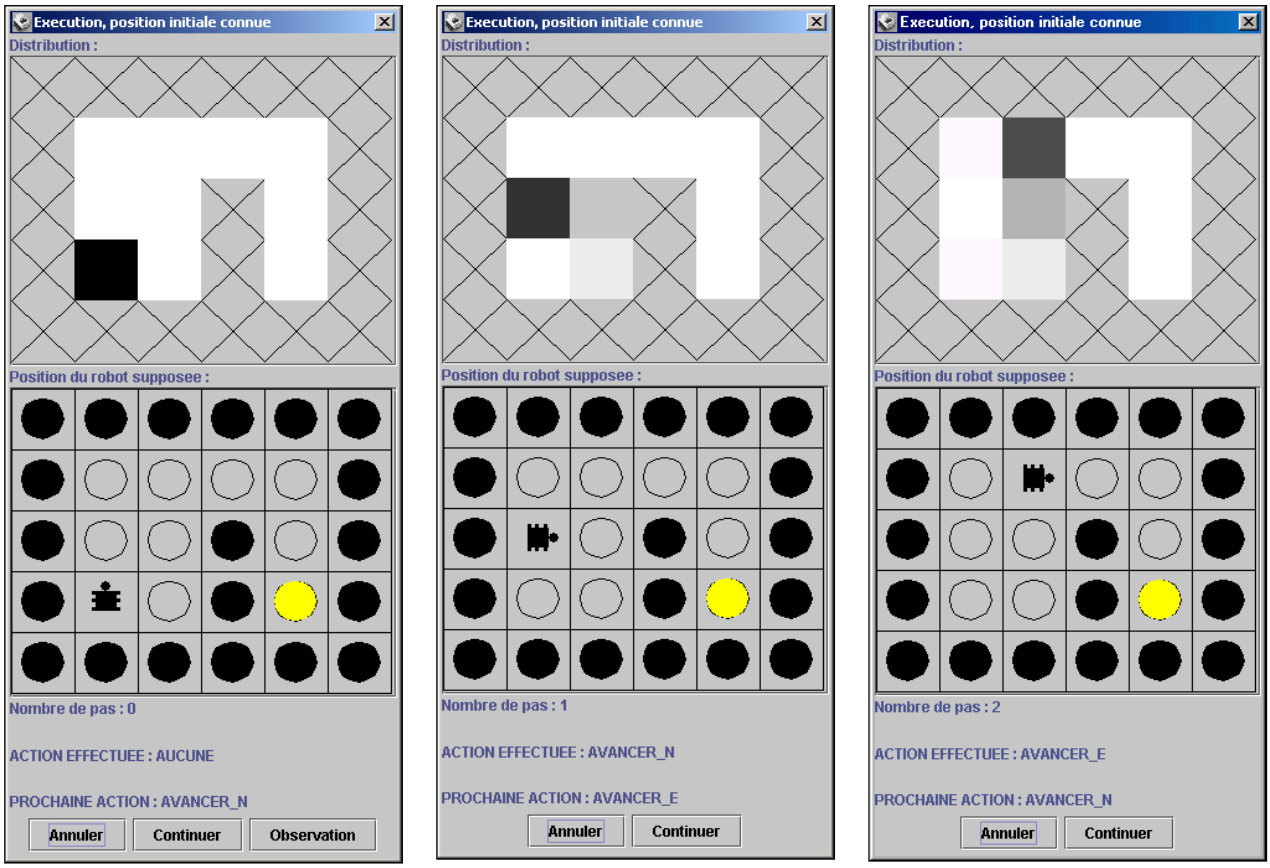
Figure 4.3 *– Le plan de l'environnement de la figure 3.1*

Nous allons par la suite décrire le déroulement d'une exécution, dans le cas où la position initiale du robot serait connue.

Dans l'interface graphique que nous avons construit pour l'exécution, il y a deux cadres. Dans le premier cadre, nous indiquons la distribution, autrement dit les états probables dans lesquels le robot peut se trouver (plus le carré est foncé, plus forte est la probabilité que le robot se trouve dans cette case). Et, le deuxième cadre est dédié à la position dans laquelle nous supposons que le robot se trouve, position très influencée par la distribution.

Pour l'exemple de la figure 4.3, nous choisissons comme position initiale du robot, la ligne 3 et la colonne 1, le robot étant orienté vers le nord.

Comme nous l'avons dit auparavant, à la première itération, le robot sait avec certitude où il se trouve, donc, dans la figure 4.4a, nous pouvons voir que dans le cadre distribution, la probabilité qu'il se trouve dans la position initiale est égale à 1 (le carré est noir).



a *b* *c*

Figure 4.4 *– Les 3 premières itérations de l'exécution*

Le robot va effectuer l'action d'avancer vers le nord. Une fois après avoir effectué l'action, le robot va faire une observation. Nous avons construit deux modes pour accueillir les observations du robot : un dans lequel nous lisons les observations dans un fichier et l'autre dans lequel nous faisons une saisie manuelle avec une boîte de dialogue. Comme nous l'avons dit précédemment, quand le robot effectue une action de transition il peut se trouver dans une des cases adjacentes qui l'entoure. Initialement le robot se trouve orienté vers le nord, et parmi les cases adjacentes, comme nous pouvons le voir dans la figure 4.5, il y en a seulement trois qui ne sont pas des obstacles.

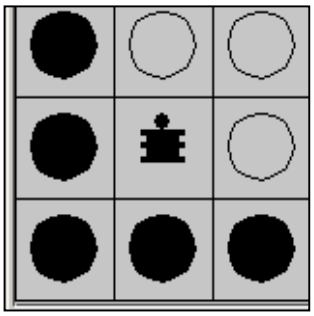
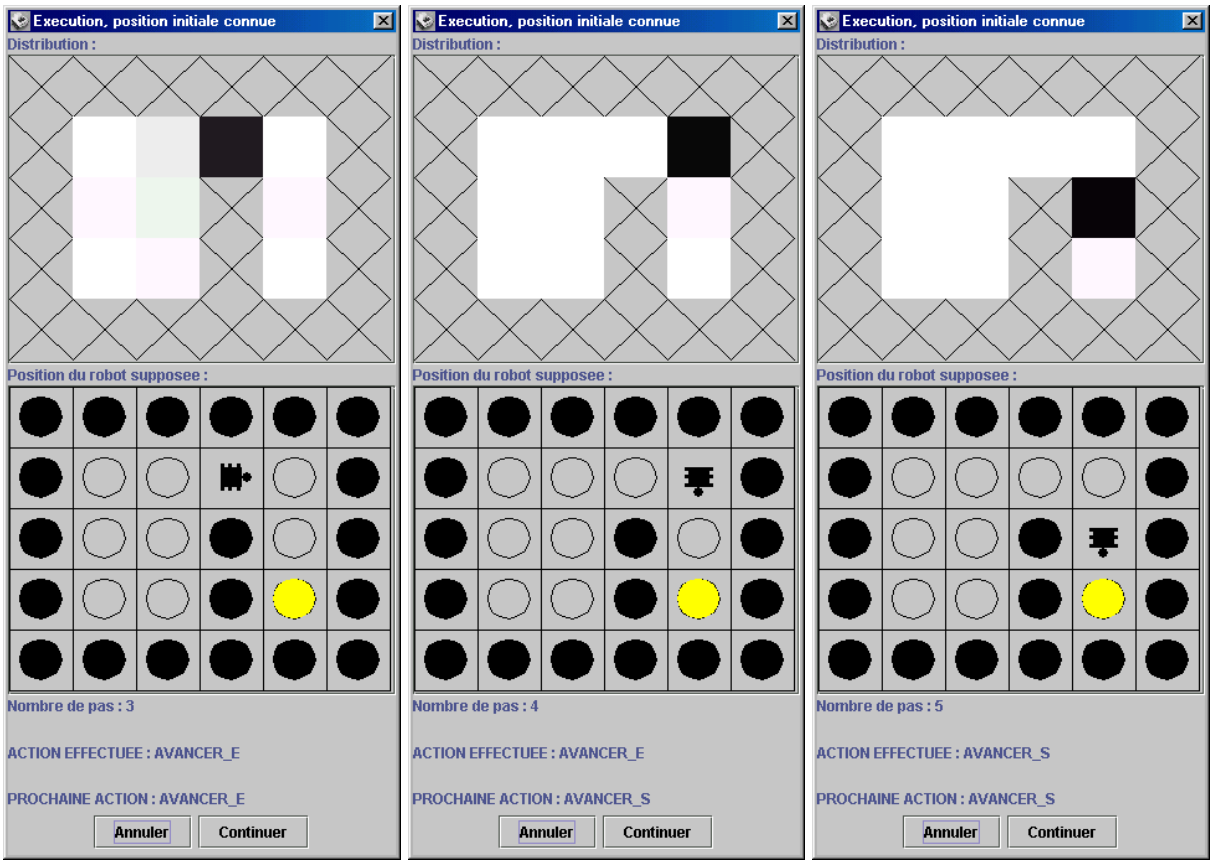


Figure 4.5 *– Zoom sur la postion du robot*

Donc en effectuant l'action avancer nord, le robot peut se trouver dans une des trois cases libres, avec une probabilité plus ou moins grande.

Dans la figure 4.4b, nous pouvons apercevoir qu'après avoir avancé et après avoir reçu les nouvelles observations, il y a trois états dans lesquels la probabilité, que le robot se trouve, est assez grande. Ainsi, la probabilité que le robot se trouve dans l'état (2,1) = 0.73, la probabilité qu'il se trouve dans l'état (2,2) = 0.21, et la probabilité qu'il se trouve dans l'état (3,2) = 0.06. Dans le cadre « Position du robot supposée », le robot est positionné sur la case où la probabilité est maximale, donc dans la case (2,1).

Mais là où cela devient intéressant, c'est ce que nous pouvons voir dans la figure 4.4c. Le robot se trouvant dans la position (2,1) comme nous avons vu précédemment, va effectuer l'action avancer à l'est. Après avoir exécuté l'action, le robot va faire les observations qui vont l'aider à se repérer dans l'espace. Le robot voit (obstacle, obstacle, libre, obstacle, libre). Donc, après avoir mis à jour la distribution, nous trouvons une forte probabilité que le robot se trouve dans l'état (1,2). Alors, nous pouvons tirer la conclusion qu'à cause d'un glissement des roues (par exemple le sol était mouillé) le robot arrive dans l'état (1,2) au lieu d'arriver dans l'état (2,2). Nous ne pouvons pas prévoir ce type de comportement dès la planification.



a *b* *c*

Figure 4.6 *– Les trois itérations suivantes*

Dans le cas illustré dans la figure 4.6a, le robot a fait des observations faussées à cause des ses capteurs bruités, et donc au lieu de voir (obstacle, obstacle, libre, libre, obstacle) il a vu (obstacle, obstacle, libre, obstacle, obstacle). Comme la distribution tient compte des positions probables précédentes (la distribution au temps t-1), nous trouvons une probabilité très forte que le robot se trouve dans l'état (1,3).

Pour les autres itérations, nous suivons le même principe et nous obtenons les résultats exposés dans les figures 4.6b,c, 4.7.

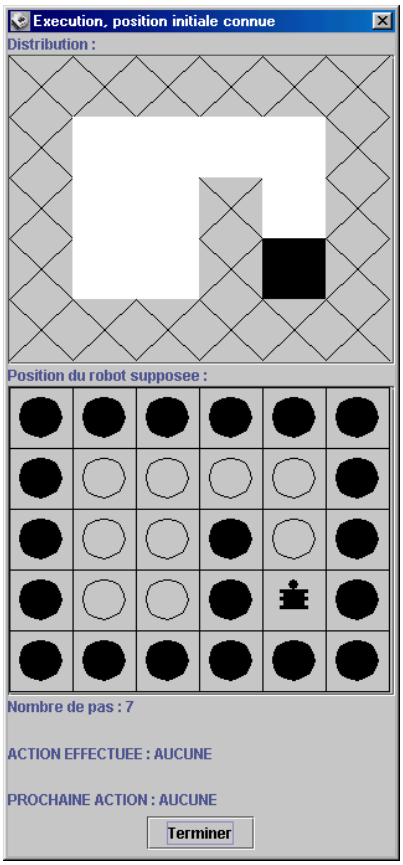


Figure 4.7 *– La dernière itération de l'exécution*

Une fois arrivé sur le but, la probabilité que le robot se trouve sur le but est maximale, donc l'exécution est terminée et nous affichons une fenêtre, qui montre les états par lesquels le robot est passé (voir figure 4.8).

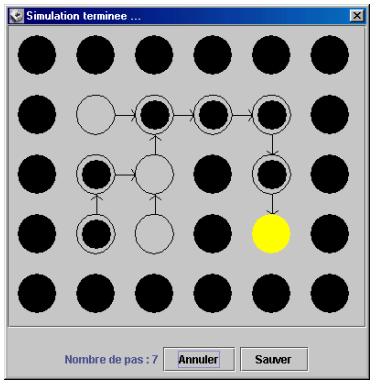
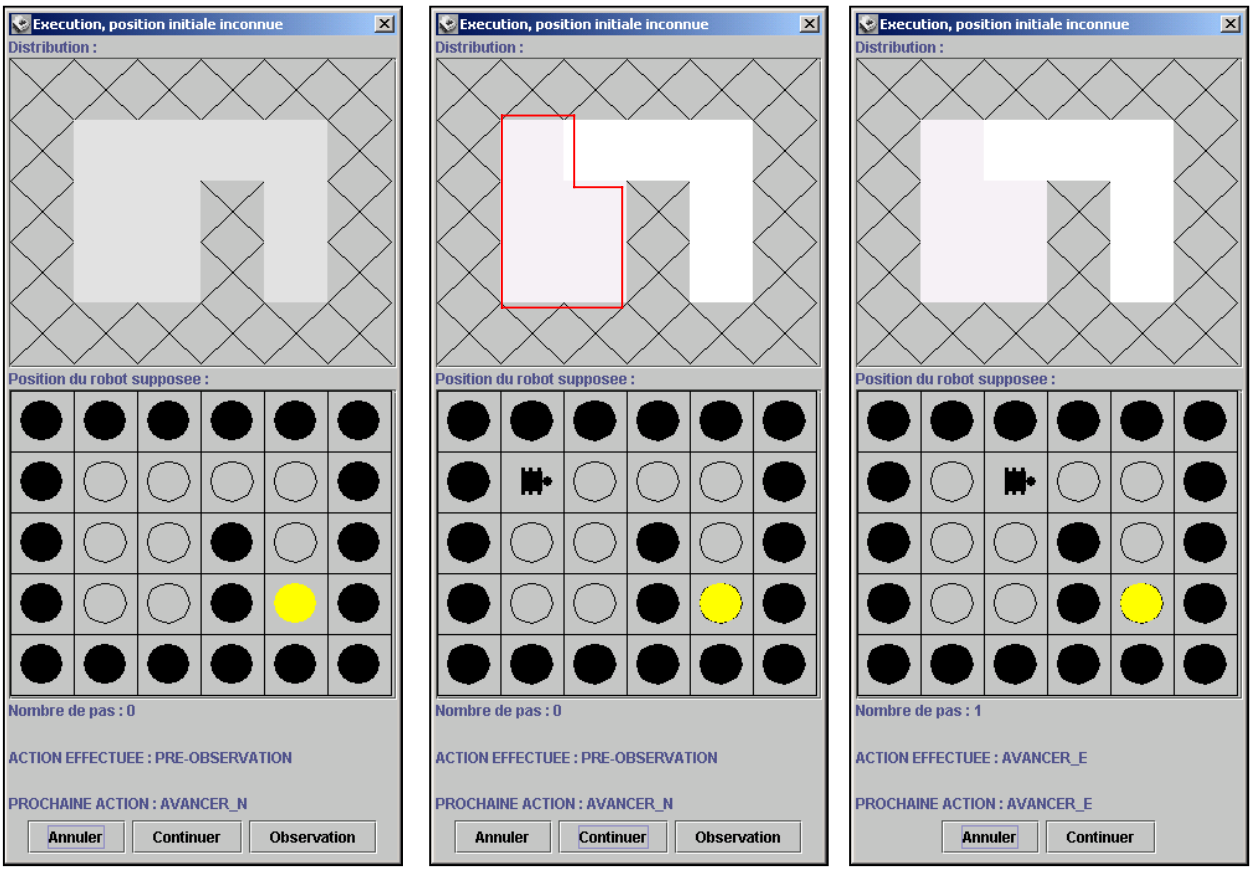


Figure 4.8 *– Le chemin parcouru par le robot*

Position initiale inconnue

Quand la position initiale du robot est inconnue, les probabilités de chaque état s de l'environnement sont équiprobables : pour tout s $Pr_{t=0}(s) = 1/|S|$.

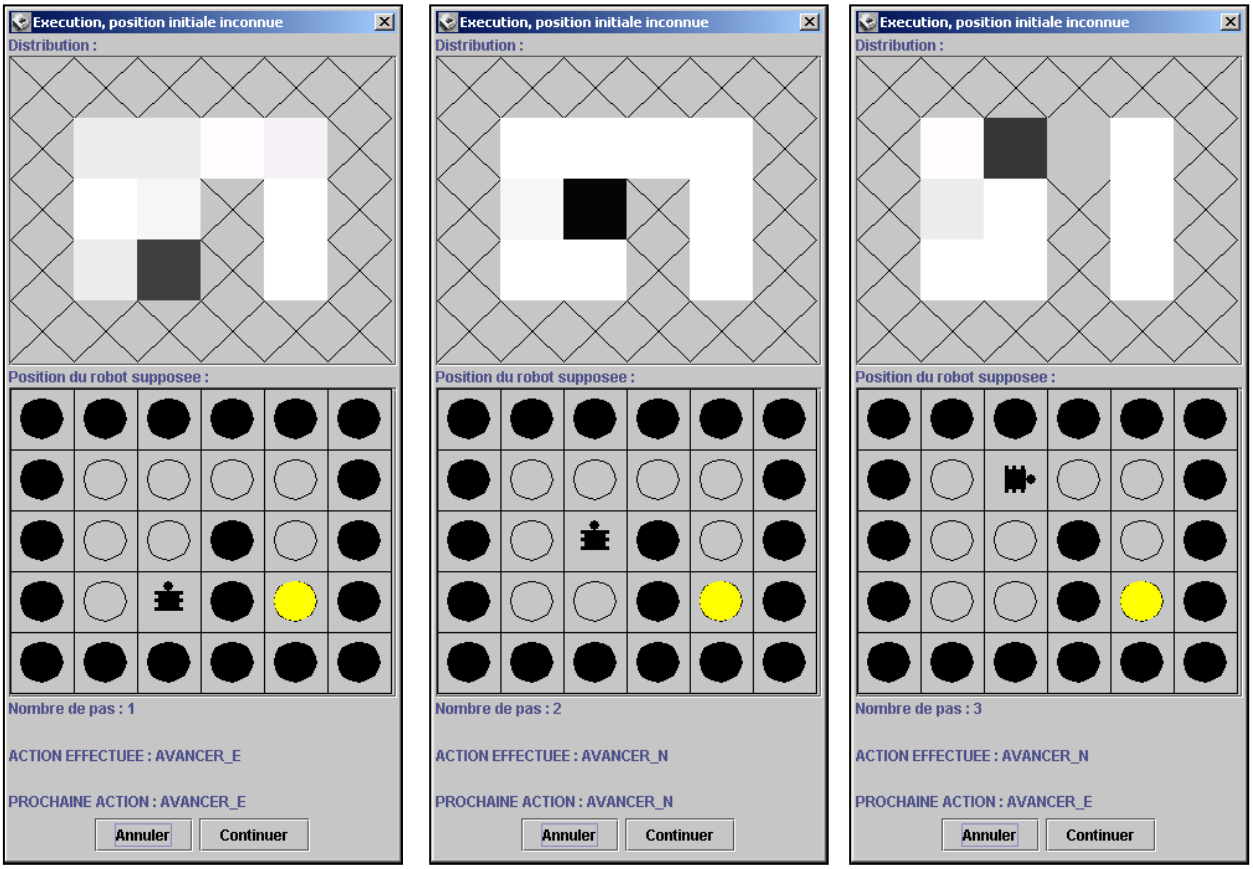
Nous pouvons voir cela dans la figure 4.9a, dans le cadre « Distribution », la probabilité de chaque état est égale à 1/10 = 0.1, 10 étant le nombre d'états libres. Comme la position initiale du robot est inconnue, avant de localiser le robot quelque part sur la carte, il faut recueillir les observations du robot, qui vont nous donner une approximation de sa localisation. Le robot envoie comme observations : obstacle à sa droite, obstacle dans la position avant droite, et libre dans toutes les trois autres positions, devant, avant gauche et gauche. En fonction de ces observations, nous trouvons cinq états (qui sont entourés dans le cadre rouge sur la figure 4.9b) dans lesquels le robot peut probablement être avec la même probabilité et nous affichons comme position supposée du robot, la première position trouvée (1,1) (voir figure 4.9b).



a *b* *c*

Figure 4.9 *– Les premières itérations de l'exécution*

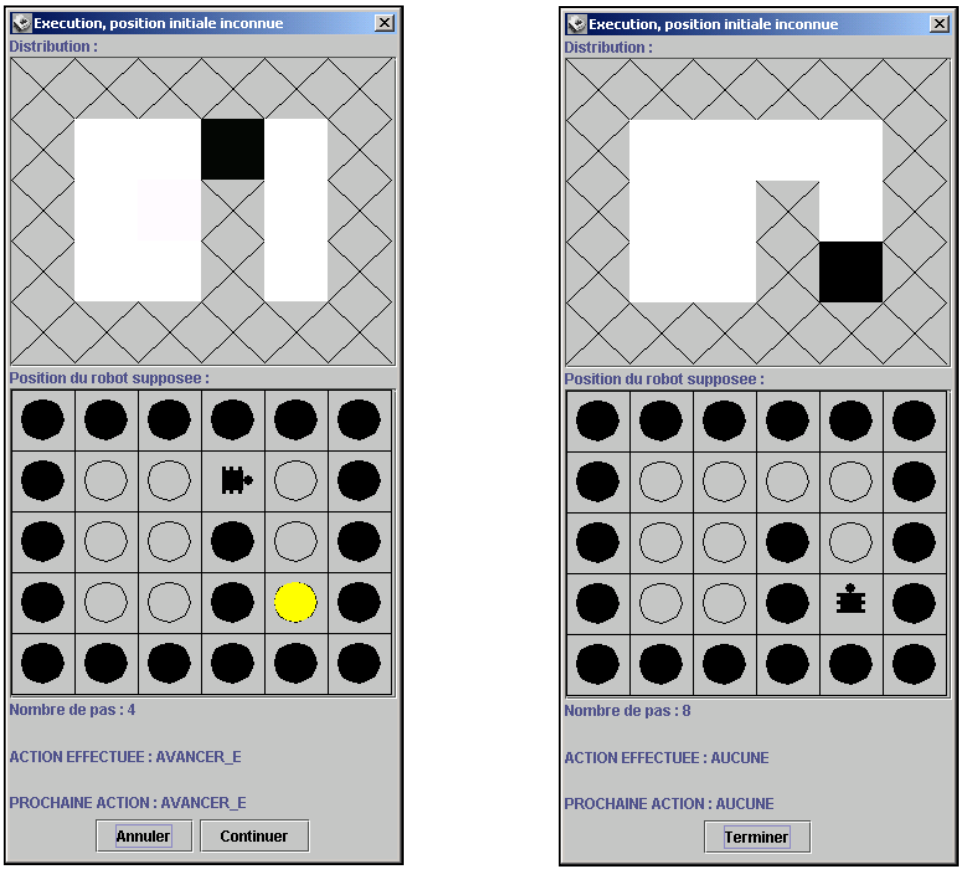
Au pas suivant le robot effectue l'action d'avancer à l'est. Après avoir effectué cette action, le robot va observer libre à sa gauche et obstacle dans toutes les autres positions. Quand le robot va se localiser de nouveau, il va se rendre compte que la position supposée précédemment était fausse et il va arriver à se localiser, après seulement deux itérations (voir figure 4.10a).



a *b* *c*

Figure 4.10 *– Les itérations suivantes*

Le déroulement de l'exécution se poursuit de la même manière que pour le cas où la position initiale du robot est connue. Nous pouvons même dire que le cas où la position initiale est connue est un cas particulier du cas où la position initiale du robot est inconnue. L'algorithme de localisation est le même dans les deux cas. Le « plus » présent dans le cas où la position initiale est inconnue, est qu'il y a une pré-observation qui nous aide à donner une approximation de la position initiale du robot.



a b

Figure 4.11 *– Les dernières itérations*

4.7. Conclusion

Dans ce chapitre, nous avons présenté une technique appelée localisation Markovienne qui calcule au fur et à mesure de l'exécution du plan une distribution des probabilités sur les états de l'environnement. Nous avons également présenté une stratégie de choix de l'action en fonction de la distribution de la probabilité sur les états.

Dans le cadre de l'exécution, nous avons deux cas possibles : le premier lorsque la position du robot est inconnue et le second cas est lorsque cette position est connue. Néanmoins, l'algorithme reste le même dans les deux cas. Nous pouvons même dire que le cas de la position connue du robot est un cas particulier de la position inconnue. 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.

Un autre des grands avantages est que même lorsque le robot se perd, il arrive à se repérer. Et cela grâce à une observation discriminante qu'il va rencontrer après avoir exécuté plusieurs actions.

Cependant, nous pouvons regretter de ne pas avoir eu le temps de tester l'exécution d'un plan par le robot Koala à l'aide de notre logiciel.

Dans la partie suivante, nous allons illustrer un problème plus complexe qui est la recherche d'un but de position inconnue.

5. Recherche d'un but de position inconnue

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 d'atteindre un but 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.

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 extraite de notre prototype.

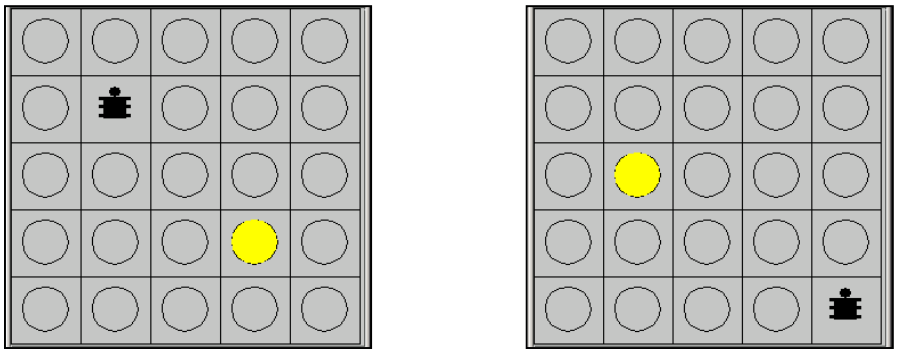
5.2. Exposé de la méthode

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 premier 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 dans certains cas d'atteindre le but plus rapidement.

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

Toutes les informations dont nous disposons sont les observations du robot et quelques données qui sont transmises par les sensors 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, c'est-à-dire 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 un axe avec les origines dans les coordonnées du robot, axe, orienté vers le bas (voir Figure 5.2).

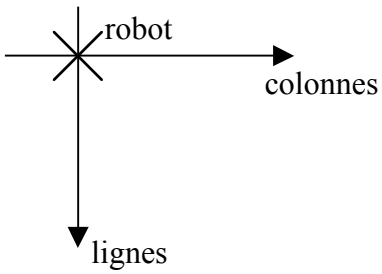


Figure 5.2 *– Axe pour la localisation du but par rapport au robot*

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 sensor du robot. Donc, nous allons mettre sur la carte autant de buts qu'il y a d'hypothèses possibles. Les hypothèses seront les états dans lesquels le robot peut probablement se trouver. Nous allons avoir une propagation des buts.

Si dans la planification (chapitre 3), nous avons présenté la fonction de gain comme :

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

Dans ce cas là, nous allons changer la fonction de gain. Les obstacles vont avoir une récompense de $\frac{-5}{1-\gamma}$ et les états libres une récompense égale à -5. Nous avons multiplié par un facteur 5 pour avoir des buts très attirants (forte probabilité) et d'autres peu attirants (faible probabilité).

Soit (x_s,y_s) les données du sensor du robot. Si le robot se trouve dans un état (x,y) avec une probabilité p, la récompense que nous donnons à l'état (x_s+x, y_s+y) pour que celui-ci soit un but est de (p – 1) * 5. Donc nous allons avoir plusieurs buts avec des récompenses différentes. La distribution sur les buts sera déterminée grâce à la fonction de gain R, car si R ≈ 0, il y a une grande probabilité que ça soit un but et si R ≈ -5 la probabilité que ça soit un but est très faible.

Donc, nous pouvons modéliser de cette manière les buts, en donnant une récompense plus grande aux buts les plus probables.

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. La distribution et les informations sur le but vont nous aider à calculer une nouvelle distribution sur les buts.

Nous allons faire une itération de l'algorithme de planification *Policy Iteration,* avec les connaissances que nous détenons sur la distribution sur les buts. Alors pendant la phase de planification, nous allons éliminer les buts qui ont une récompense très proche de –5 (récompense faible) et nous allons garder les buts avec la récompense la plus proche de 0 (récompense forte). Lorsqu'un but est supprimé, il est remplacé par un état libre et donc il va avoir une récompense de –5.

Après le pas de planification, nous allons exécuter une action. Chaque fois que nous nous déplaçons, nous allons avoir de nouvelles informations qui vont nous dire où le but se trouve par rapport au robot.

Nous allons répéter ces deux méthodes jusqu'à ce que le robot arrive au vrai but, but qui peut-être par exemple une odeur.

Notre approche apporte quelque chose de nouveau. Les deux méthodes que nous allons effectuer en parallèle, ont déjà été étudiées auparavant, la planification a été exposée dans le cadre du chapitre 3 et l'exécution d'une politique a été étudiée en détail dans le chapitre 4.

Examinons cette approche dans les paragraphes suivants, à l'aide d'exemples bien détaillés.

5.3. Résultats

Pour illustrer le fonctionnement de notre approche, nous allons prendre l'environnement présenté dans la figure 5.3. Cet environnement comporte une pièce au centre avec une seule entrée et plusieurs couloirs, dont deux à droite de la pièce, séparés par une cloison (un mur).

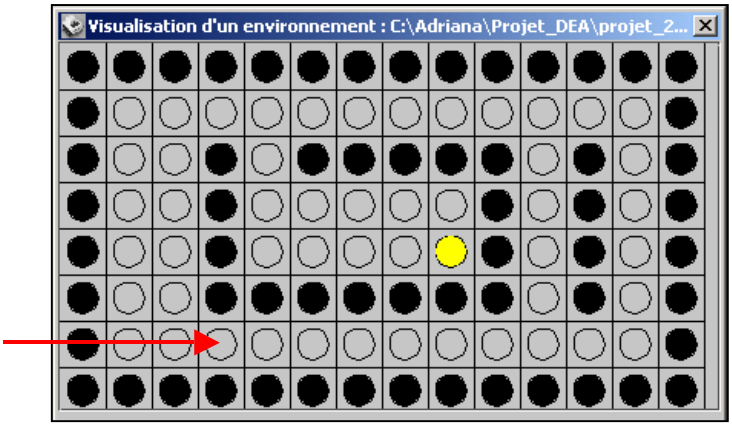


Figure 5.3 *– Environnement simple (51 états)*

Avant de commencer à détailler l'exemple, indiquons les paramètres dont nous allons avoir besoin. Pour la planification, au début, à l'itération 0, nous allons fixer la valeur du facteur _ à 0.99, car ce paramètre sert à pondérer la valeur des gains futurs.

Pour mieux suivre la description de l'exemple, nous donnons à titre informatif la position et la direction réelle du robot. Ainsi, le robot se trouve dans la case correspondant à la ligne 6 et à la colonne 3, avec l'orientation à l'ouest (voir flèche rouge sur la figure 5.3).

A la première itération, après avoir recueilli les observations (1, 1, 0, 0, 1), et les données du sensors (-2 lignes, 5 colonnes), les buts correspondants ont été propagés sur la carte.

Une fois que les buts ont été disposés sur la carte, nous effectuons une planification, qui nous construit les différents chemins qui mènent aux buts.

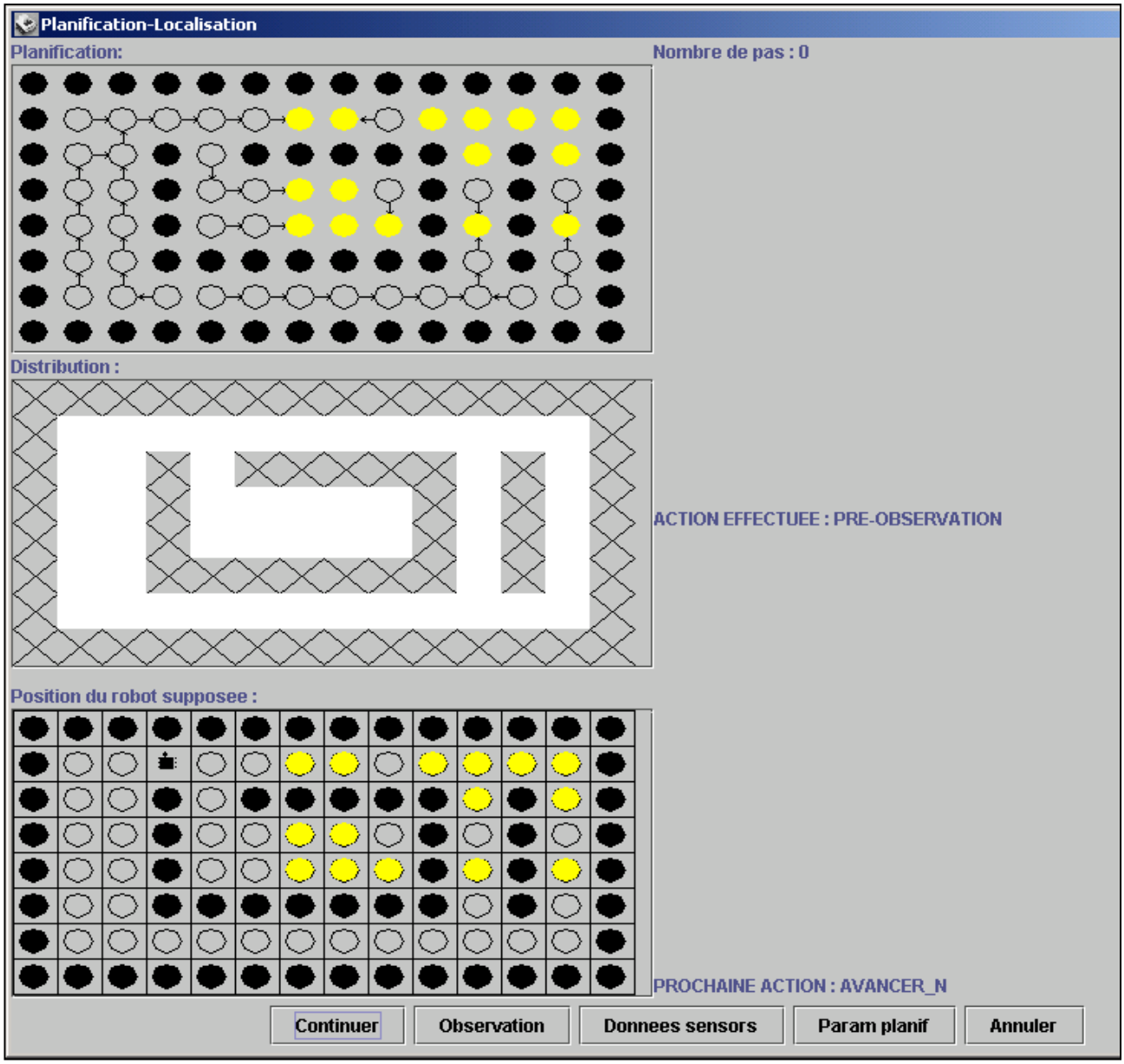
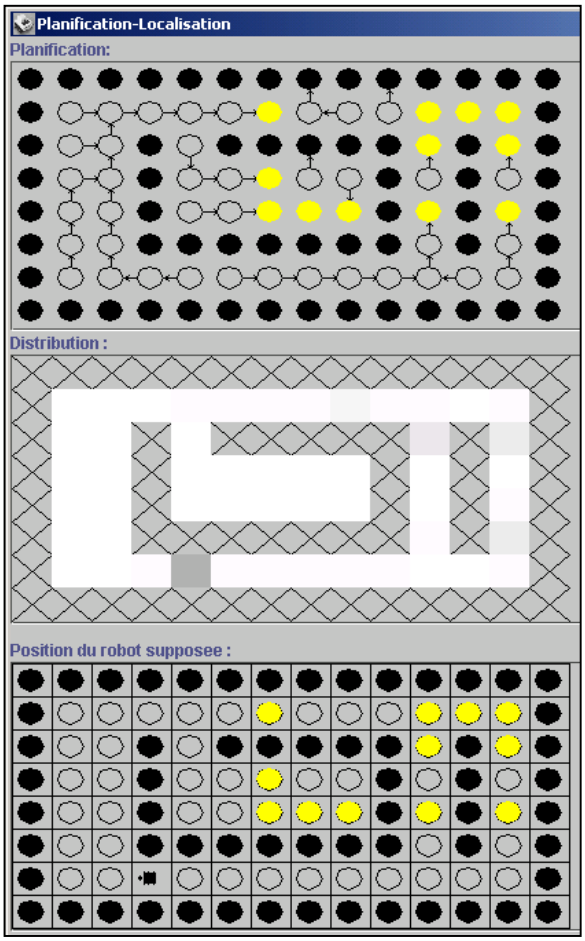


Figure 5.4 *– L'itération 0 de la planification – localisation*

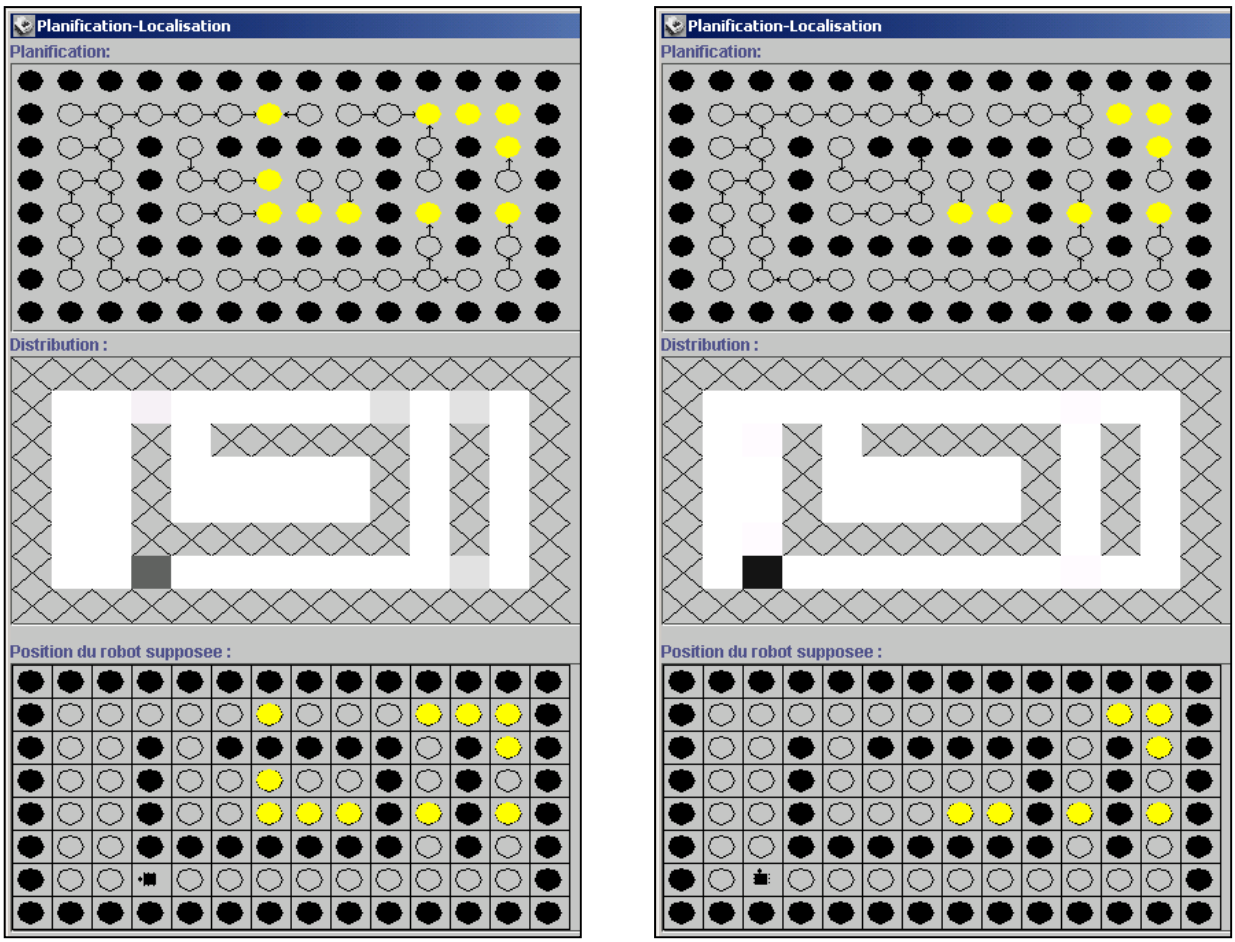
A l'itération 1, nous éliminons en premier les buts qui correspondent à des positions peu probables du robot et nous replanifions, par la suite. Les états buts supprimés deviennent des états libres, et donc avec une récompense de –5. A la planification, les actions que nous allons effectuer dans ces états, vont nous rapprocher du but le plus proche. Le robot prend les observations et les données des sensors, et il fait exécuter une nouvelle politique, en tenant compte de tous les paramètres changés.

Même si au début, la position supposée du robot était fausse ; en exécutant l'action d'avancer à l'est, et le fait de recueillir une observation discriminante, lui a permis de se relocaliser dans l'espace.



Etape 1

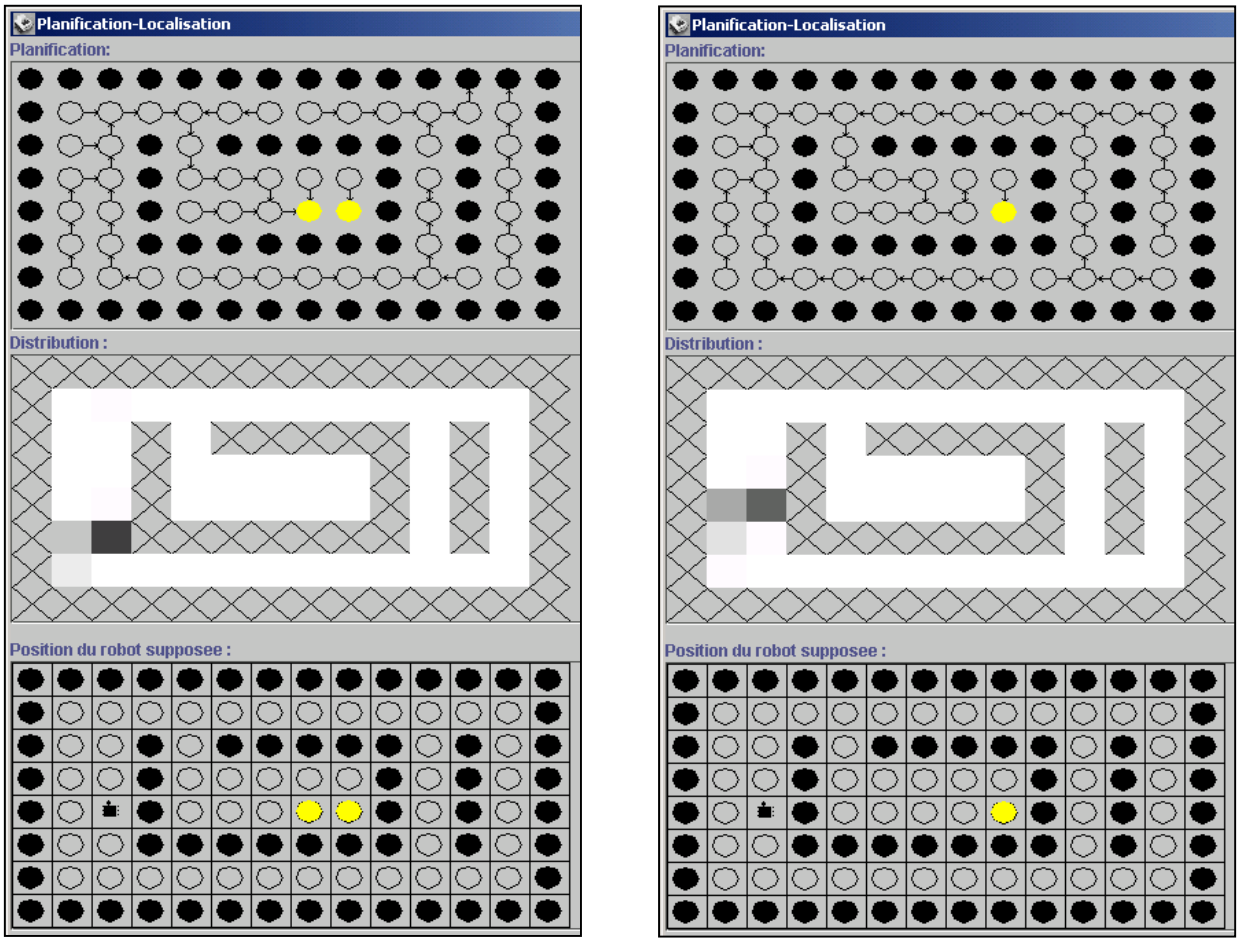
Figure 5.4 *– Itération 1 de la planification – localisation*



Etape 2 *Etape 3*

Figure 5.5 *– Les itérations suivantes*

L'enchaînement des étapes est toujours le même, comme nous pouvons le voir dans la figure 5.5. Au fur et à mesure des observations et des données sensors du robot, le nombre de buts décroît. La planification est réalisée au fur et à mesure, à chaque pas.



Etape 4 *Etape 5*

***Figure 5.6** – Améliorations*

Comme nous pouvons bien l'observer dans la figure 5.6 (étape 4), dans l'environnement ne sont restés uniquement que les deux buts les plus probables. Pourquoi ?

Nous donnons les coordonnées relatives du but par rapport au robot : (-1, 6) lorsque le robot est en (5, 2) juste avant l'étape 4. La distribution (voir fig. 5.6 – étape 4) nous indique que le robot est soit en (5,2) (plus probable et donc carré plus foncé), soit en (5, 1), soit en (6, 1), mais le but serait alors un obstacle dans ce cas-là, ce qui est bien sûr impossible. Puisque le robot a des chances de se situer sur ces 2 emplacements, il ne reste donc plus que 2 buts.

Lors de l'étape 5, la distribution est actualisée, et le robot se rend alors compte que le but qui se trouve (4, 7) a une récompense bien moins élevée que le but se trouvant en (4, 8). Par conséquent, le premier but est éliminé.

Lorsqu'il ne reste plus qu'un seul but, nous effectuons alors la planification et l'exécution comme d'habitude.

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 : un pas de planification, un pas d'exécution, 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.

Comme nous l'avons dit précédemment, nous regrettons de ne pas avoir pu tester notre simulateur avec le robot Koala. Ce dernier point fait finalement parti de la liste des extensions que nous allons mettre en place et que nous exposons par la suite.

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 aimerions développer.

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. En effet, si nous utilisons des MDP, c'est pour obtenir des politiques dont l'exécution est bien sûr rapide, mais également pour obtenir des politiques sûres. Résoudre un MDP pour obtenir une politique dont le seul critère d'optimalité est la distance n'a aucun intérêt. Nous montrons donc 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, n'hésitant 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 _ 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 décrivons les méthodes d'exécution d'une politique. Nous allons introduire des différents modules qui vont gérer la localisation du robot. Dans le cadre de l'exécution, nous avons deux cas possibles : le premier lorsque la position du robot est inconnue et le second cas est lorsque cette position est connue. Néanmoins, l'algorithme reste le même dans les deux cas. Nous pouvons même dire que le cas de la position connue du robot est un cas particulier de la position inconnue. 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.

- Dans le chapitre 5, nous présentons nos travaux qui abordent une nouvelle approche. Il s'agit, pour le robot d'atteindre un but 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 une nouvelle distribution qui ensemble avec l'information qu'il dispose sur le but, va nous aider de calculer une distribution sur les buts possibles. Cette distribution va être calculée grâce à la fonction de gain. Nous allons faire une itération de policy itération avec les informations que nous disposons sur la distribution sur les buts. 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.

Le seul regret se situe dans le fait de ne pas avoir pu tester directement notre application sur le robot ce qui me tenait à cœur au début de ce stage.

6.2. Perspectives

Nous avons présenté les principales perspectives de ce travail en fin des différents chapitres. Nous les reprenons ici en y ajoutant des perspectives que nous n'avons pas encore évoquées précédemment.

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 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 stratègie 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 la montre la figure 4.1, dont nous n'en tenons pas compte dans notre application. 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. Extensions au niveau graphique

Enfin, de nombreuses extensions au niveau graphique s'avèrent très intéressantes. Notre prototype permet à l'utilisateur d'observer le déroulement du calcul d'un plan et celui d'une exécution pas à pas. Aussi, pour insister sur le côté pédagogique de notre logiciel, nous pouvons intégrer la capacité de revenir sur le pas précédent ou même plusieurs pas en arrière. Cependant, cette option semble au premier abord très difficile à mettre en place car elle nécessite de stocker chaque politique pour le calcul d'un plan, ainsi que chaque distribution et position du robot pour une exécution.

6.2.5. 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.

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.

Annexe 1

Description du robot

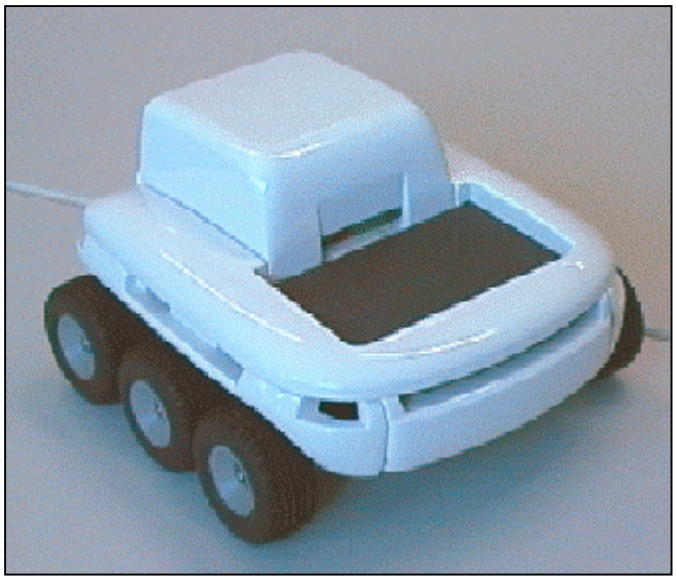


Photo du Koala

Quelques caractéristiques du Koala :

- Taille : Longueur : 32 cm
  Largeur : 32 cm
  Hauteur : 20 cm.
- Poids : environ 3 kilos.
- Capteurs : 16 capteurs infrarouges sensibles à la proximité et à la lumière.
- Autonomie : environ 3 heures
- Mouvement : 2 DC moteurs gèrent respectivement les mouvements des trois roues situées de chaque côté du robot.

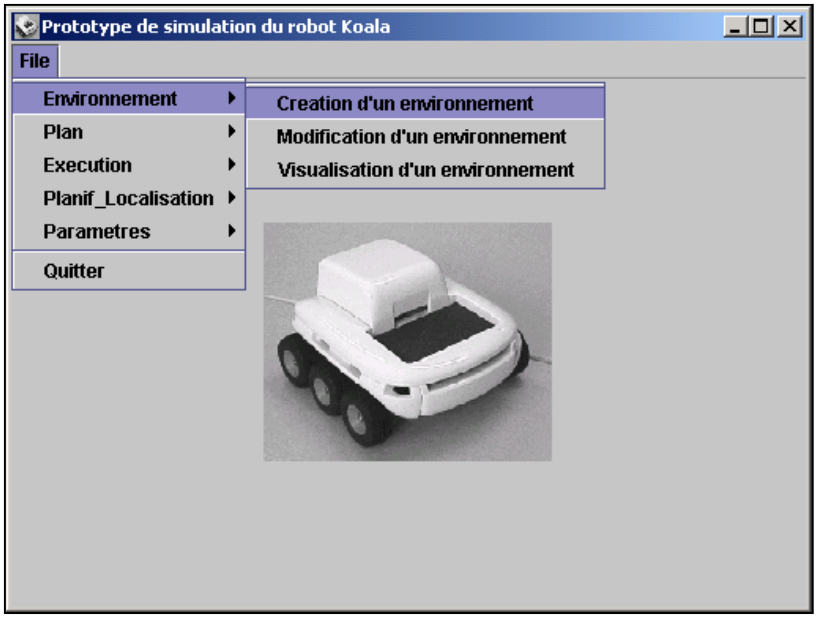
Annexe 2

Guide utilisateur

Le logiciel est disponible sur le site Web du laboratoire Gravir/INRIA.

Le prototype permet de modéliser différents environnements constitués d'obstacles et de buts, de calculer différents types de planification à partir de ceux-ci, de lancer une exécution suivant un plan et enfin de faire une planification localisation en parallèle.

En lançant l'application, une fenêtre principale présente à travers un menu les fonctionnalités du logiciel. Cette documentation expose l'utilisation de ces options dans le même ordre que le menu.



1 Environnement

1.1 Création d'un environnement

Il est possible de créer un environnement en indiquant le nombre de lignes et le nombre de colonnes de l'environnement souhaité. La détermination de ces paramètres se fait au travers de la fenêtre suivante qui s'affiche automatiquement :

Une fois les paramètres saisis, il suffit de valider pour pouvoir créer l'environnement de manière interactive, à l'aide de la grille des états de l'environnement. En effet, en cliquant sur une case, la couleur du cercle change et par-là même la nature de l'état également. Le changement de la nature se fait de manière cyclique et initialement, tous les états sont libres (couleur de fond).

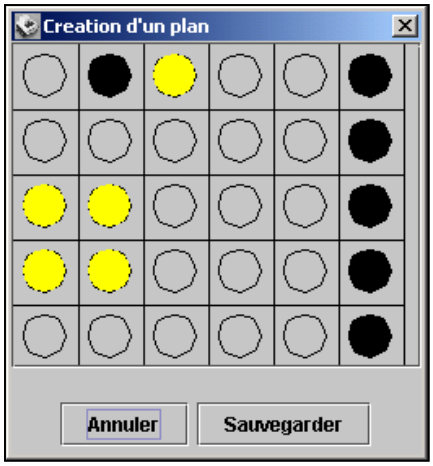
Par un clic sur un état libre, l'état devient un obstacle (noir).

Par un clic sur un état obstacle, l'état devient un but (jaune).

Par un clic sur un état but, l'état devient un état libre (couleur de fond).

Pour plus de maniabilité, il est possible de modifier la nature de plusieurs états en même temps. Ainsi, le clic sur une première case et le lâchement du bouton souris sur une autre case modifie la nature de tous les états situés dans le rectangle de diagonale formée par les deux cases.

Les deux figures suivantes exposent l'environnement initial pour la première et les possibilités de création pour la seconde.



Lorsque l'environnement souhaité est modélisé, l'utilisateur doit valider celui-ci et ensuite déterminer son nom pour le sauvegarder dans un fichier afin de le réutiliser par la suite. Tous les fichiers gérés par l'application sont stockés dans le répertoire /test_env.

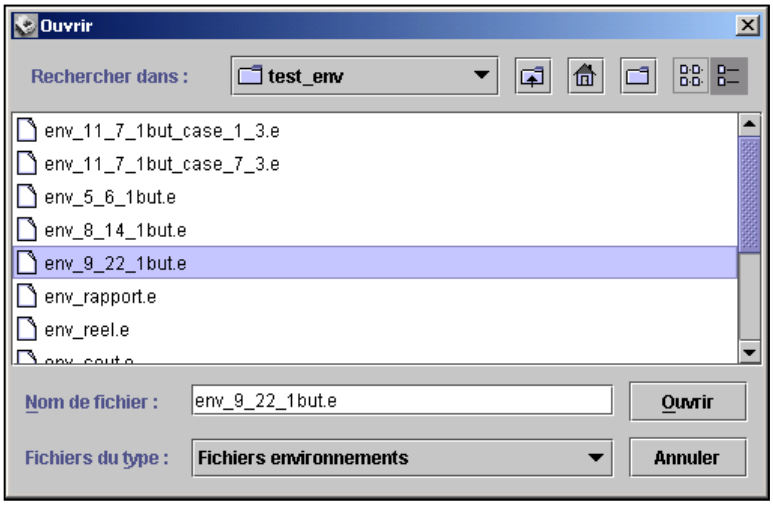
L'ensemble des noms des fichiers environnements doit être suffixé par « .e » indiquant ainsi sa nature et définissant de manière unique ce type de fichier.

Il est possible de renoncer l'environnement en cliquant sur Annuler ou de le sauvegarder en appuyant sur Sauvegarder.

1.2 Modification d'un environnement

Cette fonctionnalité permet de récupérer un environnement déjà créé pour ensuite lui apporter des modifications au niveau de la nature de ses états uniquement mais pas au niveau de sa taille.

Ainsi, l'utilisateur doit donc sélectionner le nom du fichier environnement voulu parmi ceux disponibles et ensuite le charger en cliquant sur Open.



Par la suite, l'environnement sélectionné est affiché dans une fenêtre et comme lors d'une création d'un environnement, l'utilisateur peut le modifier de la même manière et ensuite cliquer sur Valider pour sauvegarder le nouvel environnement sous le même nom ou non.

1.3 Visualisation d'un environnement

Cette fonctionnalité permet uniquement de visualiser un environnement créé ! Aucune modification ne peut être faite !

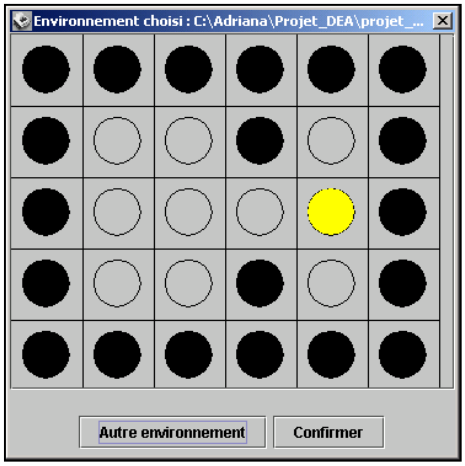
L'utilisateur doit comme pour l'option précédente sélectionner un environnement parmi les fichiers environnements existants.

2 Plans

Cette partie du logiciel permet à partir des environnements créés de générer des plans suivant plusieurs algorithmes.

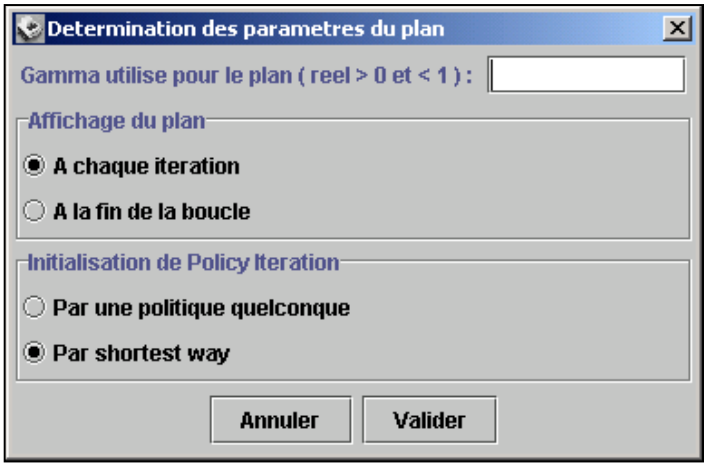
Pour les trois premières options de cette partie permettant de déterminer l'algorithme de calcul, l'utilisateur doit charger l'environnement dont il veut calculer un certain plan, de la même façon que précédemment.

Après avoir cliqué sur Open, l'environnement choisi est affiché afin que l'utilisateur confirme son choix ou revienne dessus.



Ensuite, suivant l'algorithme de planification choisi, des paramètres sont nécessaires au calcul du plan.

Ainsi, pour les algorithmes Value Iteration et Policy Iteration, il faut déterminer la valeur de _ et le mode d'affichage désiré.



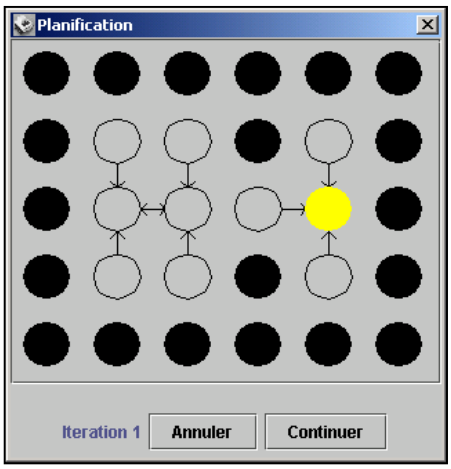
De plus, pour Policy Iteration, il faut également déterminer l'initialisation de l'algorithme soit par une politique aléatoire soit par la politique du plus court chemin.

Par contre, l'algorithme du plus court chemin ne nécessite aucun paramètre et est donc affiché directement.

Pour les deux autres algorithmes, deux modes d'affichage sont disponibles :

- à chaque itération
- une fois le calcul du plan terminé

Le premier mode d'affichage permet à l'utilisateur de contrôler le déroulement de l'exécution de l'algorithme itératif et doit donc appuyer sur le bouton Continuer pour poursuivre le calcul.

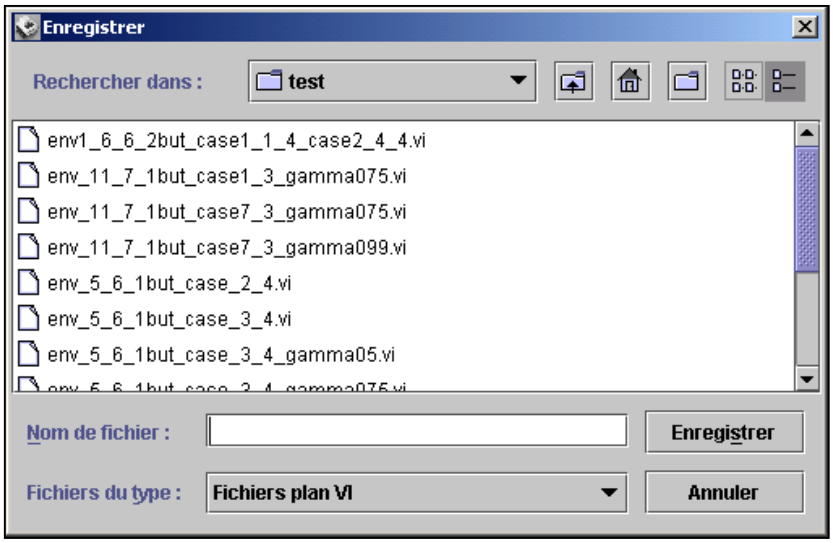


Le second mode d'affichage ne donne pas de contrôle à l'utilisateur, si ce n'est au début où il doit confirmer ou annuler sa démarche, et affiche le plan seulement une fois l'algorithme terminé.

Après chaque planification, le plan obtenu est donc affiché et l'utilisateur peut alors sauvegarder ce dernier en lui donnant un nom de fichier. L'utilisateur devra ajouter au nom de son plan un suffixe déterminant le type de planification :

- .vi pour Value Iteration
- .pi pour Policy Iteration

- .sw pour l'algorithme du plus court chemin



Enfin l'utilisateur peut visualiser un plan calculé avec la dernière fonctionnalité, en sélectionnant le fichier correspondant.

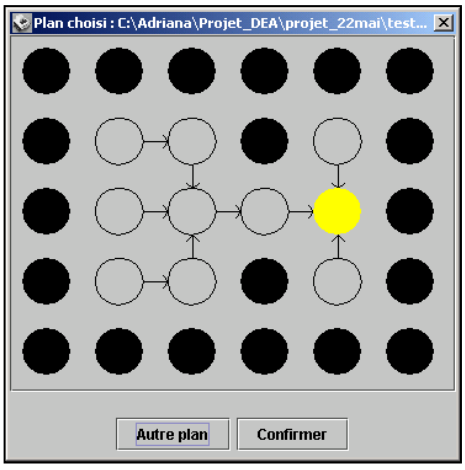
3 Exécution d'une politique

Cette partie du logiciel permet d'exécuter une politique par le robot.

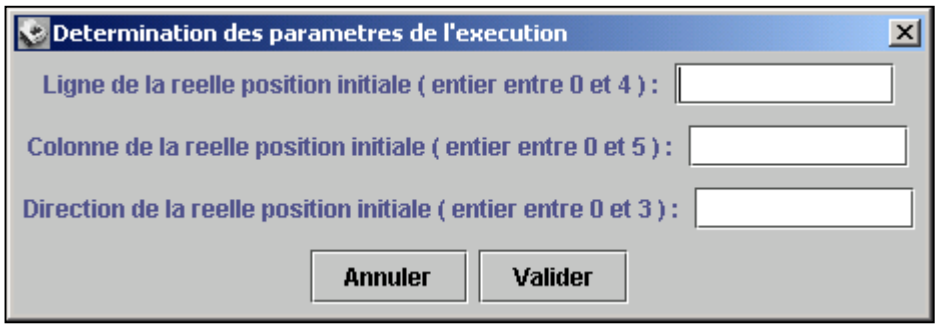
Il y a deux options déterminant si la position initiale est connue ou non. Si la position initiale est connue, l'utilisateur devra la déterminer par la suite sinon aucun paramètre n'est nécessaire.

Et une fois le but atteint, pour le mode position initiale inconnue, il n'y a plus de trace du test car nous ne connaissons pas la position réelle du robot lors de l'exécution.

Dans les deux cas, l'utilisateur doit choisir un plan sur lequel l'exécution sera basée et peut revenir sur son choix en cliquant sur le bouton autre plan, comme après l'affichage de celui-ci.



Si nous nous trouvons dans le cas où la position initiale est connue, l'utilisateur doit déterminer la position initiale du robot. Aussi, si l'utilisateur donne les coordonnées d'un état obstacle, une erreur sera levée !

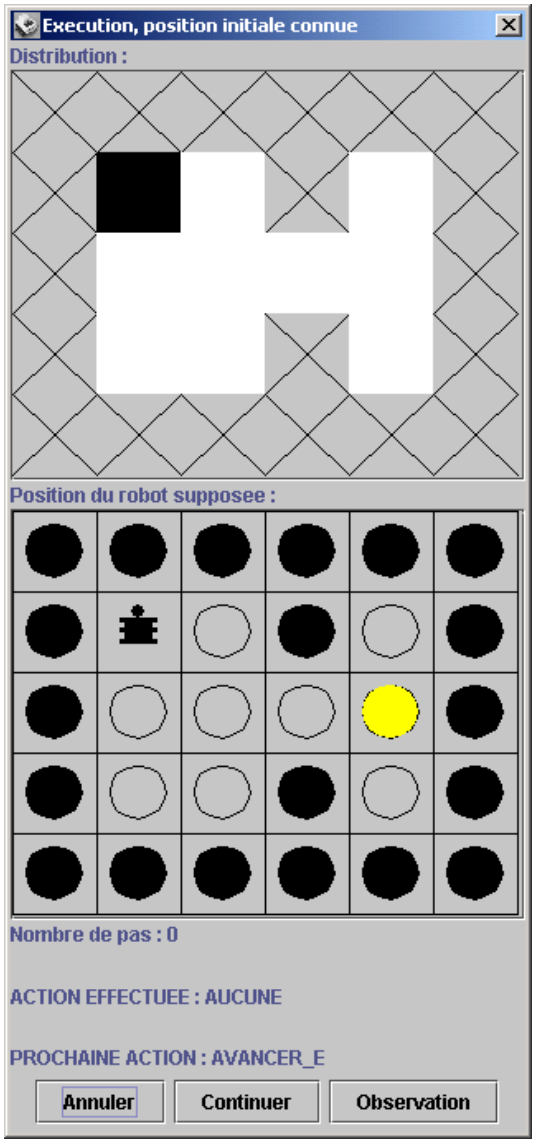


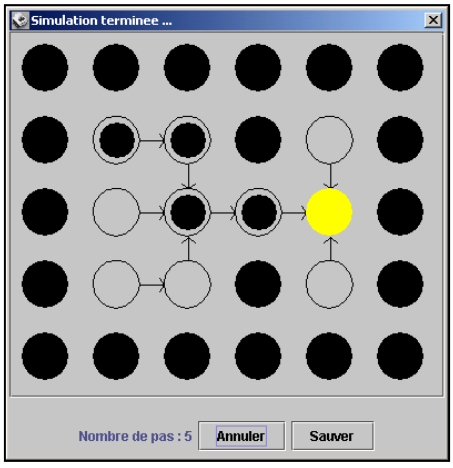
Pour les deux types d'exécutions, avant de commencer il va falloir cliquer sur Observation, qui va nous donner le choix entre une saisie manuelle ou une saisie dans un fichier d'observations du robot.

Comme pour le déroulement d'une planification, l'utilisateur doit cliquer sur Continuer pour poursuivre l'exécution.

Chaque fois que nous cliquons sur Continuer, et si au début nous avons choisi que les observations du robot soient prises d'un fichier, les données vont être lues automatiquement du fichier ; sinon une fenêtre va s'ouvrir et nous allons pouvoir faire une saisie manuelle des observations du robot.

Le premier panneau indique la distribution des états, c'est à dire que plus le carré correspondant à un état a une couleur sombre, plus la probabilité que le robot s'y trouve est grande. Le dernier panneau indique la position où le robot est supposé se trouver étant donné la distribution des états et la stratégie de localisation choisie.





Une fois le but atteint et la probabilité de s'y trouver assez grande, la trace de l'exécution est affichée et l'utilisateur peut la sauvegarder à l'aide du suffixe .sc.

4 Planification Localisation

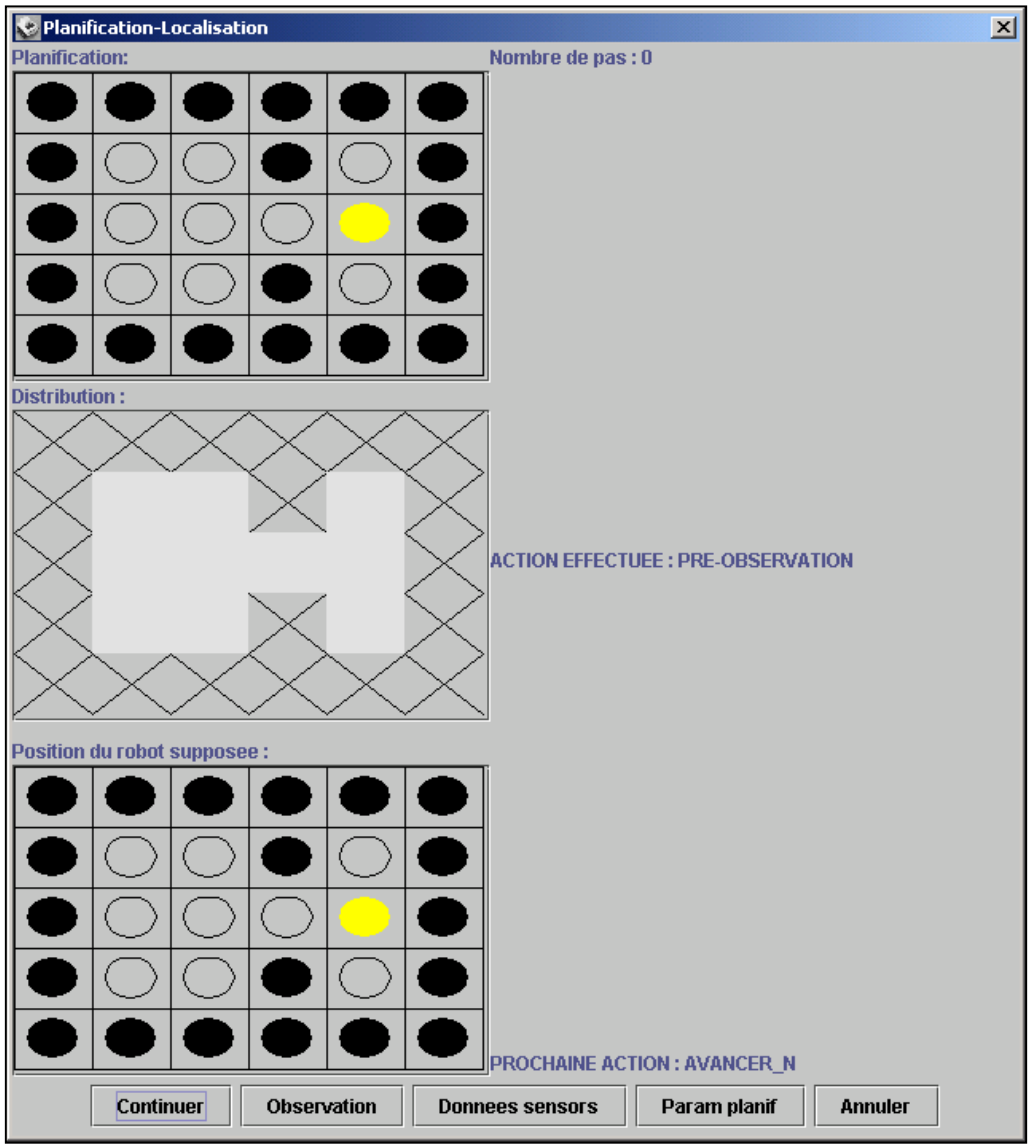
Cette partie est un parallélisme entre la planification et l'exécution. A la première itération il y a plusieurs paramètres qui sont nécessaires pour la suite des calculs. Ainsi il faut déterminer la valeur de _ pour la planification, et si nous prenons les données du sensor et les observations du robot dans un fichier ou avec une saisie manuelle.

Le premier panneau indique la planification.

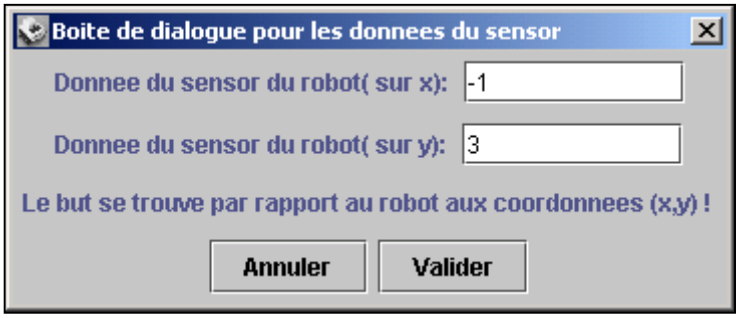
Le deuxième panneau indique la distribution des états, c'est à dire que plus le carré correspondant à un état, a une couleur sombre, plus la probabilité que le robot s'y trouve est grande.

Le dernier panneau indique la position où le robot est supposé se trouver étant donné la distribution des états et la stratégie de localisation choisie.

Comme pour le déroulement d'une planification et d'une exécution, l'utilisateur doit cliquer sur Continuer pour poursuivre les calculs.



Si nous avons choisi d'accueillir les données du sensor avec une saisie manuelle, la seule chose que nous avons en plus par rapport à la planification et à l'exécution, est une boîte de dialogue qui nous interroge sur les coordonnées du but par rapport au robot :



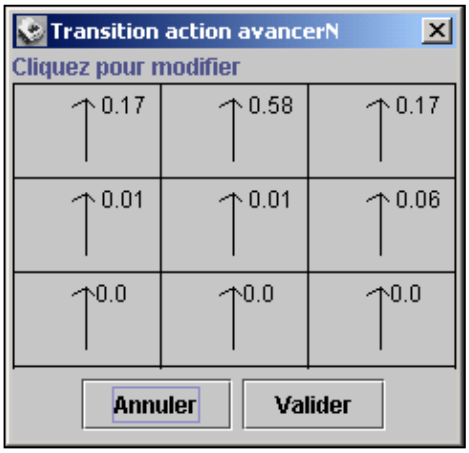
5 Paramètres

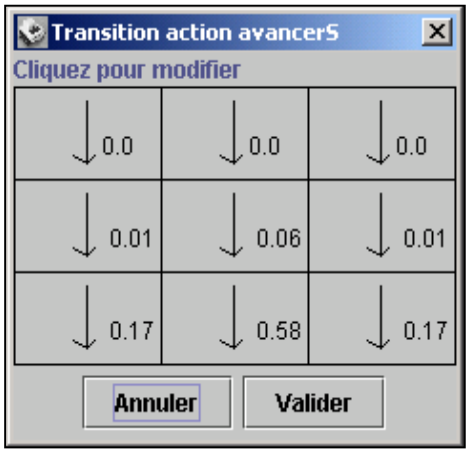
Cette dernière partie permet d'observer et de modifier certains paramètres de l'application.

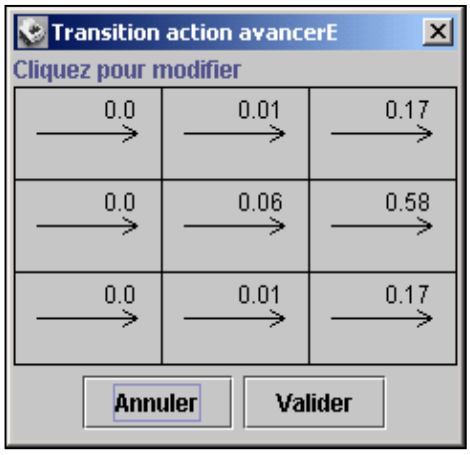
5.1 Fonction de transition

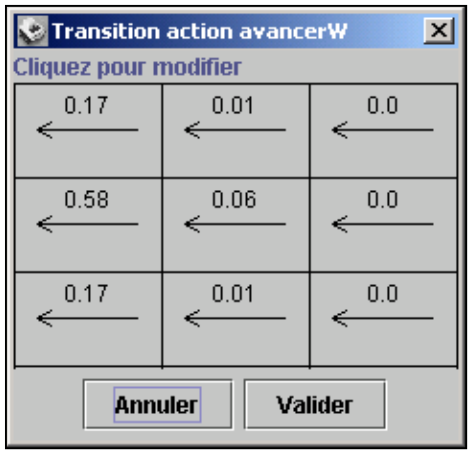
En sélectionnant cette fonctionnalité, une fenêtre demande à l'utilisateur de choisir l'action, dont il veut obtenir ou modifier la fonction de transition.

Ensuite, la fonction de transition de l'action choisie est affichée avec pour état initial l'état au centre, et les valeurs des probabilités de finir dans un état sont indiquées sur les états correspondants. Pour modifier ces valeurs, il suffit de cliquer sur la case et alors une fenêtre de saisie des nouvelles probabilités concernées est affichée.



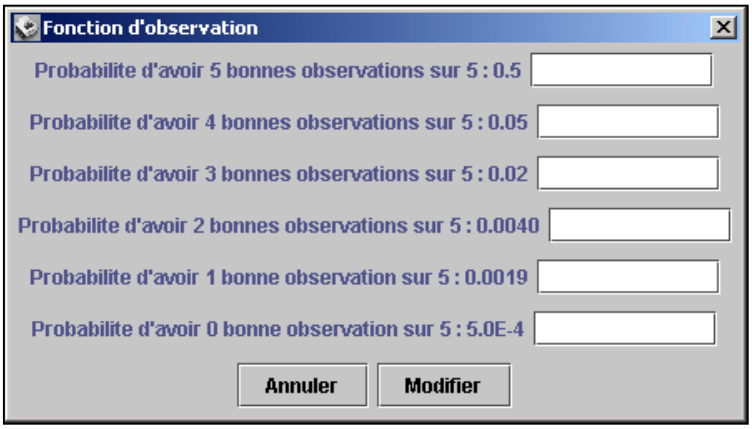






5.2 Fonction d'observation

Les probabilités de la fonction d'observation sont affichées et suivies d'une zone de saisie à l'aide desquelles l'utilisateur peut modifier ces valeurs.



6 Quitter

Cette dernière fonctionnalité permet à l'utilisateur de quitter l'application !