Titre: Title:	Commande d'un robot mobile par instructions sémantiques
Auteur: Author:	Jacques Michiels
Date:	2015
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Michiels, J. (2015). Commande d'un robot mobile par instructions sémantiques [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2043/

UNIVERSITÉ DE MONTRÉAL

COMMANDE D'UN ROBOT MOBILE PAR INSTRUCTIONS SÉMANTIQUES

JACQUES MICHIELS
DÉPARTEMENT DE GÉNIE ÉLECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE ÉLECTRIQUE)
DÉCEMBRE 2015

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

COMMANDE D'UN ROBOT MOBILE PAR INSTRUCTIONS SÉMANTIQUES

présenté par : MICHIELS Jacques
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. MALAHMÉ Roland, Ph. D., président
M. LE NY Jérôme, Ph. D., membre et directeur de recherche
M. SAUSSIÉ David, Ph. D., membre et codirecteur de recherche
M. GOURDEAU Richard, Ph. D., membre

RÉSUMÉ

Lorsqu'une personne égarée nous demande son chemin, on lui indique généralement, d'une part, une description géométrique approximative du trajet jusqu'à destination et, d'autre part, des éléments de l'environnement identifiables pouvant lui servir de repères. Les futures actions de cette personne sont alors l'objet d'un dilemme : progresser lentement et identifier avec certitude tous les repères ? Ou progresser plus rapidement vers la destination au risque de se tromper de chemin ?

Le travail réalisé dans le cadre de cette maîtrise a pour objectif de développer un algorithme permettant à un robot mobile de présenter un comportement similaire à celui d'une telle personne égarée. Nous définissons une version simplifiée du problème. Comme *éléments identifiables*, nous plaçons dans l'environnement une série d'objets que le robot est capable de détecter et d'identifier à l'aide d'une caméra et d'un algorithme de détection. Par analogie à la *description géométrique du trajet*, une série d'instructions donnent approximativement la distance et la direction de chaque objet par rapport au précédent. L'objectif du robot est alors d'atteindre le dernier objet.

Nous formulons ce problème simplifié sous forme d'un *processus décisionnel de Markov*, en nous basant sur des modèles stochastiques de l'algorithme de détection et des instructions. Nous appliquons finalement la méthode de *Monte Carlo Tree Search* à la résolution de ce problème.

ABSTRACT

When somebody is lost and asking for directions, we usually, on the one hand, provide an approximate geometric description of the itinerary towards his destination and, on the other hand, describe identifiable elements in the environment that can be used as landmarks. Then, on his way, this lost person is facing a dilemma : should he progress slowly and try and identify with certainty all the specified landmarks? Or should he move faster, hence taking the risk of taking a wrong turn?

In this master's thesis, we present an algorithm allowing a mobile robot to behave in a similar fashion to such a lost person. We define a simplified version of this problem. As *identifiable elements*, a sequence of objects are placed in the environment, which the robot can then detect and identify with the help of a camera and a detection algorithm. The *geometric description of the itinerary* is given as a sequence of instructions, which specify approximately the distance and direction of each object from the previous one. The goal of the robot is to reach the last object.

We formulate this simplified problem as a *Markov Decision Process*, based on stochastic models of the detection algorithm and instructions. The *Monte Carlo Tree Search* method is eventually used to solve this problem.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ALGORITHMES

NOTATIONS MATHÉMATIQUES

Variable aléatoire	A, B, C
Réalisation correspondante	a, b, c
Distribution de probabilité	$P(A)$
Probabilité d'une réalisation	$P(A = a) \equiv P(a)$
Probabilité booléenne	$P(A = 1) \equiv P(A) \equiv P(a)$ $P(A = 0) \equiv P(\neg A) \equiv P(\neg a)$
Ensemble	$\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$
Fonction	$\mathrm{F}(x)$, $\mathrm{G}(x)$, $\mathrm{Z}(x)$
Ensemble vrai	$\mathcal{F} \triangleq \left\{ x \;\middle|\; \mathrm{F}(x) = 1 \right\}$
Ensemble faux	$\mathcal{F}^C \triangleq \left\{ x \;\middle|\; \mathrm{F}(x) = 0 \right\}$
Vecteur, matrice	$\mathbf{v}$, $\mathbf{m}$
Composante	$\mathbf{v}_n$, $\mathbf{m}_{i,j}$
Ensemble de composantes	$\mathbf{v}_{l:k} \triangleq \left\{ \mathbf{v}_n \;\middle|\; n \in [l, k] \right\}$

CHAPITRE 1 INTRODUCTION

Pour une meilleure mise en contexte, commençons cette introduction par un exemple. Imaginons un étudiant perdu dans les couloirs de l'université à la recherche de sa salle de cours. Un guide peut alors lui décrire le chemin à parcourir. Par exemple :

> *« Avance jusqu'au bout du couloir et prends la porte sur ta gauche. Traverse ensuite la cafétéria et prends le couloir sur ta gauche juste après les machines distributrices. Ton local de cours sera alors sur ta gauche environ dix mètres après le local 101. »*

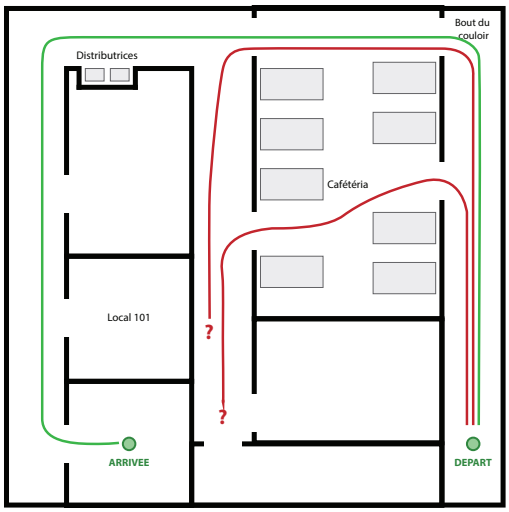


Figure 1.1 **Un étudiant perdu dans les couloirs**. En vert, le chemin correct décrit par le guide. En rouge, des chemins erronés.

Le guide et l'étudiant perdu construisent tous deux, dans leur tête, leur propre représentation de l'environnement. Pour faire le lien entre ces deux représentations, une telle description de l'itinéraire fait appel à un trait commun au guide et à l'étudiant : leur capacité d'observation. Distinguons donc deux types d'informations dans cette description :

1. **Description géométrique de l'itinéraire** : *« en face de toi »*, *« à gauche »*, *« environ dix mètres »*. Ces indications décrivent approximativement la trajectoire à parcourir dans l'environnement tel que perçu par le guide.

2. **Description de l'environnement** : *« couloir »*, *« cafétéria »*, *« machines distributrices »*, *« porte »*. Ces indications décrivent comment le guide perçoit l'environnement

et permettent à l'étudiant de faire le lien avec sa propre représentation de l'environnement : ce qu'il en connaît préalablement et ce qu'il en observe en chemin.

De manière générale, l'objectif de l'étudiant perdu est donc de faire le lien entre la description de l'environnement du guide et ce qu'il en observe pour y identifier le chemin indiqué. Une fois en route, le comportement de l'étudiant fait alors l'objet d'un compromis entre deux types de décisions :

1. **Progresser vers la destination** sur le chemin le plus susceptible d'être correct, l'environnement tel que décrit par le guide semblant correspondre à ce qu'il en connaît.
2. **Explorer l'environnement** afin d'y identifier les éléments spécifiés pour s'assurer qu'il suit actuellement le bon chemin et/ou décider du chemin à suivre ensuite.

S'il est pressé, l'étudiant peut avoir envie de progresser le plus rapidement possible vers sa destination en ne jetant qu'un coup d'œil à son environnement, quitte à ne pas être certain de l'exactitude du chemin parcouru. Le risque de prendre un chemin erroné est alors important, et le temps nécessaire pour revenir sur ces pas lorsqu'il s'en rend compte peut être significatif. Au contraire, il peut également être très prudent et observer rigoureusement l'environnement pour y identifier avec certitude tous les éléments spécifiés par son guide. Le risque de faire une erreur est dès lors beaucoup plus faible, mais sa progression peut être beaucoup plus lente.

Il s'agit d'un cas typique de **dilemme entre l'exploration et l'exploitation**, étudié à travers plusieurs domaines de recherches. Par exemple, en gestion d'entreprise, il s'agit de faire un compromis entre le financement, d'une part, d'opérations rentables à court terme et, d'autre part, de projets de développement qui pourraient l'être dans le futur (Uotila et al., 2009). Du coté de la psychologie, Cohen et al. (2007) étudient les mécanismes du cerveau qui interviennent dans un tel processus de décision.

Dans ce mémoire, nous présentons une méthode permettant à un robot mobile équipé d'une caméra de présenter, pour une version simplifiée de ce problème de suivi d'instructions sémantiques, un comportement semblable à celui de l'étudiant perdu de l'exemple ci-dessus.

1.1 Éléments de la problématique

Nous pouvons distinguer plusieurs sous-problèmes distincts dans notre problématique globale. Commençons par passer en revue les capacités requises de l'étudiant égaré pour atteindre son objectif avec les instructions du guide :

1. **Interpréter ce qu'il observe dans son environnement**. Qu'il s'agisse de lieux spécifiques comme une cafétéria, de lieux plus généraux comme un couloir ou des escaliers, ou encore d'objets comme une porte ou des machine distributrices, l'étudiant perdu de notre exemple est capable d'identifier ce qu'il observe et donc de construire sa propre représentation de l'environnement.

2. **Interpréter les instructions du guide**. L'étudiant comprend les instructions du guide et est capable d'identifier dans ce qu'il observe et connaît de l'environnement les éléments auxquels elles font référence.

3. **Planifier ses actions**. L'étudiant perdu est capable de se projeter dans le futur et d'évaluer les conséquences possibles de ses actions. Il est ainsi capable de prendre une décision réfléchie sur les actions à prendre ; où se rendre et où regarder.

Dans le reste de cette section, nous passons en revue différentes méthodes dans la littérature qui permettraient à un robot mobile de présenter des capacités semblables.

1.1.1 Observation de l'environnement

Les différentes méthodes permettant à un robot mobile d'interpréter son environnement sont typiquement liées au type de capteur utilisé. Des approches bidimensionnelles analysent les images successives d'une caméra monoculaire. D'autres approches se basent sur l'analyse de nuages de points issus, par exemple, d'une caméra stéréoscopique, d'une caméra de type RGB-D (*Microsoft Kinect*, *Asus Xtion*) ou d'un détecteur laser (*Lidar*).

Pour suivre des instructions similaires à celles du guide de notre exemple introductif, le robot devrait être capable d'identifier deux types d'éléments dans son environnement : des objets (*porte*, *machine distributrices*) et des lieux (*couloir*, *cafétéria*).

Identification d'objets

De nombreuses méthodes de détection d'objets sont basées sur l'identification de points d'intérêt, caractérisés par des descripteurs locaux comme SIFT (Lowe, 1999) ou SURF (Bay et al., 2006) dans une photo, ou encore *FPFH* (Rusu et al., 2009) dans un nuage de points. Il est alors possible de déterminer si l'objet se trouve dans une photo en comparant les points d'intérêt identifiés dans celle-ci avec des points d'intérêt connus de l'objet recherché. Ce type de méthode est généralement efficace pour détecter un objet avec des caractéristiques uniques. Par exemple, Nister and Stewenius (2006) utilisent une méthode de *vocabulary tree* pour reconnaître en temps réel des pochettes de CD issues d'une grande base de données. Atanasov et al. (2014a) utilisent un capteur de distance installé au bout d'un bras robotique pour reconnaître un ensemble d'objets placés sur une table à partir d'une base de données contenant des modèles tridimensionnels de ces objets. Pour reconnaître tous ces objets rapidement, les déplacements de ce bras robotique sont déterminés par programmation dynamique.

Cependant, plutôt que d'identifier un objet unique par ses caractéristiques, le robot devrait surtout être capable d'interpréter sémantiquement ce qu'il observe. Le guide ne devrait fournir ni photo, ni modèle des objets à identifier. Par exemple s'il mentionne une porte, le robot devrait être capable d'en identifier une, indépendamment de la forme de sa poignée et de s'il s'agit d'une porte vitrée ou non ; il sait juste qu'il doit trouver une porte.

Dans ce contexte, Felzenszwalb et al. (2010) proposent une méthode de détection et classification d'objets dans une photo unique. Cette méthode est basée sur des modèles qui identifient plusieurs caractéristiques géométriques récurrentes dans les photos d'un objet donné (DPM, *Deformable Part Models*). Plus précisément, l'analyse d'une image commence par le calcul des descripteurs locaux de type HOG (*Histograms of Oriented Gradients*, Dalal and Triggs, 2005) pour différent niveaux de lissage de l'image. En comparant ces descripteurs aux modèles, un score est obtenu pour chaque région de la photo à chaque niveau de lissage. Après seuillage, des cadres correspondant aux meilleures détections sont finalement retournés (figure 1.2).

Cette méthode présente l'avantage d'un apprentissage aisé de nouveaux modèles, les positions des objets dans les photos d'apprentissage nécessitant uniquement d'être indiquées par un cadre. En robotique mobile, Atanasov et al. (2014b) utilisent avec succès cette méthode en détectant des chaises et des portes dans l'environnement du robot pour déterminer sa localisation dans une carte sémantique de l'environnement.

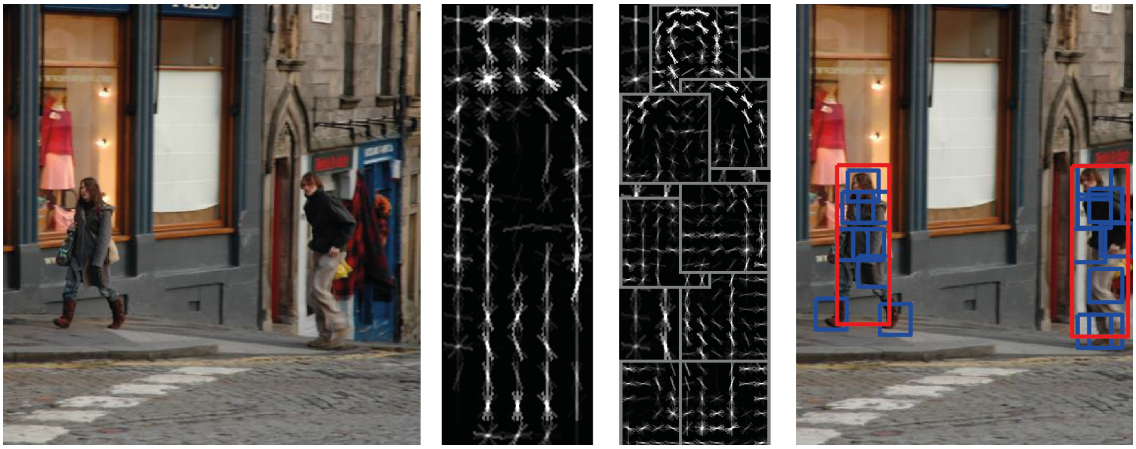


Figure 1.2 **Deformable Part Models**. Détection de personnes dans une photo par la méthode DPM. Au centre : illustration du modèle d'une *personne* basé sur des descripteurs HOG.

Identification de lieux

Pandey and Lazebnik (2011) montrent qu'une méthode basée sur les mêmes *Deformable Parts Models* (DPM), utilisés ci-dessus pour la détection et classification d'objets, peut être appliquée au problème plus général de la reconnaissance d'un lieu dans une photo. Dans ce cas, ces modèles de lieux contiennent alors des caractéristiques géométriques récurrentes, d'une part, de ce lieu (par exemple, les lignes droites d'un couloir) et, d'autre part, d'objets typiques de ce lieu (par exemple, plusieurs portes dans un couloir).

Torralba et al. (2003) proposent une méthode basée sur un descripteur global d'une photo qui permet d'identifier un lieu connu à partir d'une photo et, en outre, de catégoriser une photo d'un lieu inconnu. Ils étendent leur méthode à l'identification d'un objet en s'aidant du contexte du lieu dans lequel il se trouve. Pronobis et al. (2006) utilisent une approche similaire de descripteur global pour l'appliquer au cas plus spécifique d'un robot mobile se déplaçant dans un environnement de bureau.

1.1.2 Recherche visuelle d'un objet

Étant donné un robot mobile équipé d'une caméra, le problème de la recherche visuelle d'un objet consiste à planifier les actions du robot pour trouver cet objet le plus rapidement possible dans l'environnement.

Shubina and Tsotsos (2010) proposent une solution heuristique à un problème de ce type. Étant donné un robot mobile équipé d'une caméra directionnelle, ils cherchent à localiser un repère visuel connu dans un environnement de bureau. Le robot n'observe pas l'environnement en continu mais progresse de manière itérative : il décide d'une pose, s'y déplace, prend et analyse une photo une fois arrivé.

Avec un modèle stochastique de l'algorithme de détection, la distribution de probabilité sur la position du repère est mise à jour après chaque photo. Ils divisent l'environnement en cellules et définissent une sphère de perception (*sensed sphere)* comme l'espace géométrique visible depuis une cellule donnée. En combinant la distribution sur la position du repère à la sphère de perception et au modèle de détection, ils identifient des cellules prometteuses à partir desquelles la probabilité de détecter l'objet est grande.

Plutôt qu'une approche itérative photo par photo, d'autres méthodes de recherche visuelles sont basées sur une observation dynamique en continu de l'environnement (Meger et al., 2008). Une difficulté réside alors dans l'analyse rapide de l'important flux de données issu de la caméra. Dans ce contexte, la modélisation de l'attention visuelle (Borji and Itti, 2013), d'inspiration biologique, permet d'identifier rapidement dans un tel flux de données visuelles des régions d'intérêt à analyser.

1.1.3 Planification d'actions

La méthode de *Monte Carlo Tree Search* (MCTS) est une méthode pour la sélection d'une action optimale par la construction itérative d'un arbre, qui représente les résultats futurs possibles pour chaque action applicable. Initialement proposée par Kocsis and Szepesvári (2006), cette méthode a eu beaucoup de succès dans le domaine de l'intelligence artificielle, notamment avec son application au jeu de Go (Gelly and Silver, 2011). Browne et al. (2012) passent en revue les nombreuses variantes et applications de cette méthode.

Processus Décisionnel de Markov

L'application de la méthode MCTS passe par la formulation du problème étudié sous la forme d'un processus décisionnel de Markov (*Markov Decision Process*, *MDP*). Un MDP comprend quatre éléments (Bertsekas, 2005) :

1. Un **ensemble d'états** $\mathcal{S}$. Étant donné l'état initial s_0, nous notons $s_t \in \mathcal{S}$ l'état obtenu après t actions ; autrement dit, l'état du système à l'instant t.

2. Un **ensemble d'actions** $\mathcal{A}$. A chaque état $s \in \mathcal{S}$ du système correspond un ensemble $\mathcal{A}_s$ d'actions a applicables. Nous notons $a_t \in \mathcal{A}_{s_t}$ l'action appliquée à l'instant t.

3. Une **fonction de transition** $\mathrm{T}\,(s_{t+1}, a_t, s_t)$ donne la probabilité que le système soit dans l'état s_{t+1} à l'instant t+1 étant donné l'état s_t et l'action a_t à l'instant précédent.

4. Une **fonction de récompense** $\mathrm{R}(s_t, a_t)$ donne une récompense quantitative à l'application de l'action $a_t \in \mathcal{A}_{s_t}$ lorsque le système est dans l'état s_t.

Nous nous limiterons ici à l'étude du cas particulier des problèmes à horizon infini avec actions terminales. L'**horizon infini** signifie qu'il n'y a pas de limite prédéfinie sur le nombre d'actions successives à appliquer. La séquence d'états et d'actions pourrait ainsi continuer indéfiniment. Une **action terminale** clôture la séquence d'états et d'actions. Nous notons $\mathcal{A}_T$ l'ensemble des actions terminales.

Une **politique** $\pi : \mathcal{S} \to A$ d'un MDP est une fonction qui spécifie pour chaque état du système $s_t \in \mathcal{S}$ une action $a_t \in \mathcal{A}_{s_t}$ à appliquer. La **valeur d'un état** $v_\pi(s_t)$ d'après une politique π est la somme espérée des récompenses jusqu'à ce qu'une action terminale soit appliquée. Nous écrivons sous forme récursive :

$$v_\pi(s_t) = \mathrm{R}(s_t, \pi(s_t)) + E\Big(v_\pi(s_{t+1}) \,\Big|\, s_t, \pi(s_t) \Big)$$

où la valeur espérée du prochain état $v_\pi(s_{t+1})$ étant donné une action a_t appliquée à l'état s_t précédent est déterminée par la fonction de transition :

$$E\Big(v_\pi(s_{t+1}) \,\Big|\, s_t, a_t \Big) = \begin{cases} 0 & \text{si} \quad a_t \in \mathcal{A}_T \\ \sum\limits_{s_{t+1} \in \mathcal{S}} \Big(\mathrm{T}\,(s_{t+1}, a_t, s_t) \cdot v_\pi(s_{t+1}) \Big) & \text{sinon} \end{cases} \tag{1.1}$$

A défaut d'état ultérieur, la valeur espérée après une action terminale $a_t \in \mathcal{A}_T$ est nulle. La **valeur optimale** $v^*(s_t)$ d'un état s_t est la plus grande valeur possible de cet état :

$$v^*(s_t) = \max_{a_t \in \mathcal{A}_{s_t}} \Big[\mathrm{R}(s_t, a_t) + E\Big(v^*(s_{t+1}) \,\Big|\, s_t, a_t \Big) \Big] \tag{1.2}$$

Par extension, la **politique optimale** π^* d'un MDP est celle qui maximise la valeur de tous les états, et l'action optimale $\pi^*(s_t)$ à appliquer à un état s_t est donc :

$$\pi^*(s_t) = \operatorname*{argmax}_{a_t \in \mathcal{A}_{s_t}} \Big[\mathrm{R}(s_t, a_t) + E\Big(v^*(s_{t+1}) \,\Big|\, s_t, a_t \Big) \Big] \tag{1.3}$$

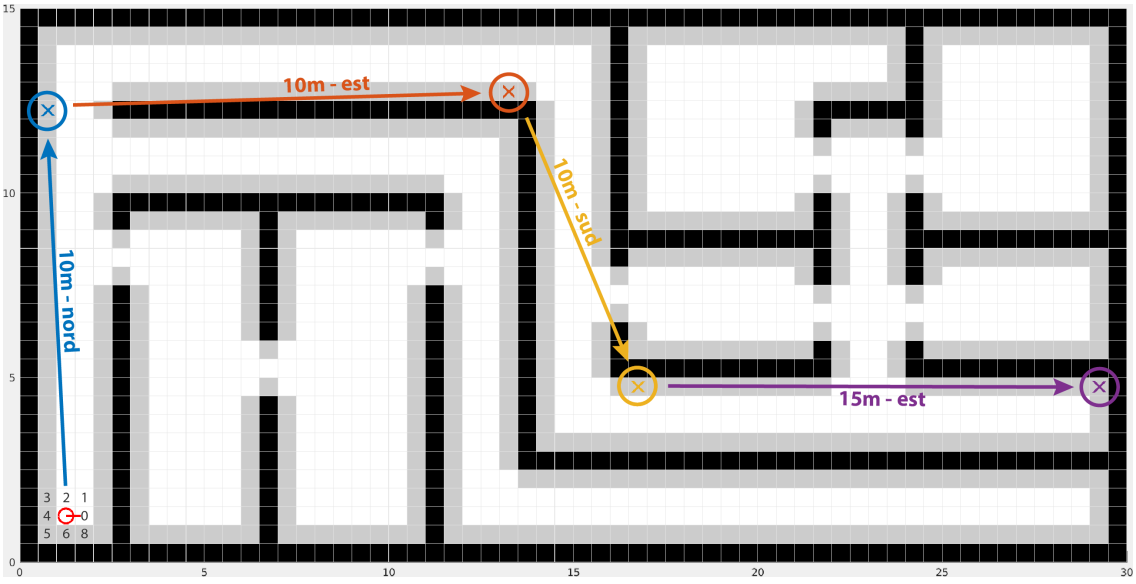


Figure 1.3 **Illustration du problème simplifié**. Le robot est représenté par le rond rouge dans le coin inférieur gauche avec une barre qui indique son orientation. Les cellules contenant les objets sont indiquées par des croix de couleurs, et les flèches indiquent les instructions successives correspondantes.

Résoudre un MDP consiste à trouver une politique aussi proche que possible de cette politique optimale. Pour des problèmes simples, une solution exacte peut parfois être obtenue. Pour des problèmes plus complexes, une solution est généralement approximée par estimation de la valeur des états subséquents.

1.2 Hypothèses du problème simplifié

Nous considérons un robot mobile équipé d'une caméra se déplaçant dans un environnement en deux dimensions dont il dispose d'une carte. Nous supposons que la localisation du robot dans cette carte est connue en tout temps à l'aide d'une méthode dont nous faisons abstraction, basée, par exemple, sur un filtrage des données issues de l'odométrie, d'un capteur laser et/ou de la caméra (odométrie visuelle). Une algorithme de détection (par exemple la méthode DPM) permet au robot de détecter, d'identifier et de localiser des objets de différentes classes (*chaise*, *porte*) dans des photos qu'il prend de l'environnement.

Comme illustré à la figure 1.3, une série d'objets identifiables sont placés dans l'environnement. Le robot ne connaît initialement pas la position de ces objets, mais une série d'instructions lui est fournie. Chaque instruction spécifie la classe d'un de ces objets (*description de*

l'environnement) et sa distance et direction approximatives par rapport à l'objet précédent, ou par rapport à la position initiale du robot pour le premier objet (*description géométrique de l'itinéraire*). L'objectif du robot est d'atteindre le dernier objet. Le robot doit dès lors résoudre le dilemme d'exploration-exploitation en décidant pour chaque objet s'il est utile d'essayer de le localiser ou s'il est préférable de passer à la recherche des objets ultérieurs.

1.3 Objectifs de recherche

Récapitulons les principales hypothèses du problème simplifié :

1. Le robot dispose d'une **carte de l'environnement**.
2. La **localisation** du robot dans cette carte est parfaite.
3. Une série d'**objets uniques** sont disposés dans l'environnement.
4. Ces objets sont identifiables par un **algorithme de détection** d'objet.
5. Pour chaque objet, une **instruction** spécifie une orientation et une distance approximative par rapport à l'objet précédent (ou à la pose initiale pour le premier objet).

Dans ce contexte, l'objectif général de ce projet est de développer un algorithme permettant à un robot mobile d'**atteindre le dernier objet** en évitant de perdre du temps ; que ce soit en se trompant de chemin, faute d'avoir localisé correctement les objets intermédiaires, ou encore en essayant le cas échéant de localiser inutilement un de ces objets intermédiaires. Nous proposons de formuler ce problème sous forme d'un processus décisionnel de Markov (MDP) et de le résoudre à l'aide de la méthode de *Monte Carlo Tree Search* (MCTS). Nos objectifs spécifiques de recherche sont alors :

1. Formuler le problème sous forme d'un **MDP** (*Markov Decision Process*).
2. Adapter la **méthode MCTS** (*Monte Carlo Tree Search*) au problème étudié.
3. Valider par **simulations** l'algorithme développé.

1.4 Plan du mémoire

Au **chapitre 2**, nous formulons mathématiquement le problème étudié et le représentons sous forme d'un MDP. Nous commençons par modéliser de manière générale le robot et son environnement. Nous définissons ensuite des modèles stochastiques sur les instructions et le détecteur d'objet. Nous finissons en utilisant ces modèles dans la définition des éléments constitutifs du MDP.

Au **chapitre 3**, nous présentons les détails de la résolution du MDP défini au chapitre précédent. Nous commençons par présenter un filtre à particules permettant une estimation rapide de l'état du système à chaque analyse d'une photo. Nous proposons ensuite une politique heuristique du MDP. Nous présentons finalement la méthode MCTS, d'abord de manière générale, puis appliquée au problème étudié.

Au **chapitre 4**, nous validons finalement la méthode développée en présentant nos résultats de simulations. Nous commençons par présenter un exemple d'évaluation des paramètres du modèle de détection d'objet. Nous commentons ensuite, étape par étape, un exemple simulé du déroulement de la recherche des objets par le robot.

CHAPITRE 2 FORMULATION DU PROBLÈME

Dans ce chapitre, nous formulons mathématiquement le problème étudié et le représentons sous forme d'un MDP. Nous commençons par modéliser de manière générale le robot et son environnement. Nous définissons ensuite des modèles stochastiques sur les instructions et le détecteur d'objet. Nous finissons en utilisant ces modèles dans la définition des éléments constitutifs du MDP. Comme ce chapitre comprend un grand nombre de définitions, nous listons dans le tableau 2.1 les principales notations des éléments du problème.

2.1 Robot et environnement

Comme illustré à la figure 1.3, nous considérons un environnement en deux dimensions divisés en n_g cellules carrées par une grille. Chaque **cellule** de cette grille peut être identifiée par une paire d'indices $(i, j) \in \mathbb{Z}^2$ spécifiant ses coordonnées discrètes. Pour plus de clarté dans les notations, nous associons à chaque cellule un identifiant unique $c \in \mathbb{N}$ et notons $(x_c, y_c) \in \mathbb{R}^2$ les coordonnées du centre de cette cellule c.

Nous définissons un ensemble de **poses discrètes** : au centre (x_c, y_c) de chaque cellule avec huit orientations possibles, faisant face à une des huit cellules adjacentes. Ainsi, chaque pose discrète est identifiée par un triplet d'indices $(i, j, d) \in \mathbb{Z}^3$ spécifiant la cellule (i, j) considérée et un indice d'orientation discrète $d \in \{0, 1 \dots 7\}$ spécifiant l'orientation $d \cdot \pi/4$ correspondante. De nouveau, nous associons un identifiant unique $p \in \mathbb{N}$ à chaque pose discrète. Nous notons $a_p \in [0; 2\pi]$ l'orientation et $(x_p, y_p) \in \mathbb{R}^2$ les coordonnées de cette pose.

Occupation et navigabilité

Comme illustré à la figure 2.1, une **carte d'occupation** $\mathrm{O} : \mathbb{N} \to \{0, 1\}$ spécifie pour chaque cellule de la grille si elle est occupée par un élément statique (typiquement, les murs), à l'exception des objets recherchés. Par extension, une **carte de navigabilité** $\mathrm{N} : \mathbb{N} \to \{0, 1\}$ spécifie pour chaque cellule de la grille si le robot peut s'y rendre. Typiquement, cette carte de navigabilité est semblable à la carte d'occupation mais prend en compte les dimensions du robot pour exclure les cellules dans lesquelles il pourrait entrer en collision avec un élément de l'environnement.

Tableau 2.1 Abrégé des notations

n_g	Nombre de cellules dans la grille
$c \in \mathbb{N}$	Identifiant d'une **cellule**
$p \in \mathbb{N}$	Identifiant d'une **pose** discrète
$(x_c, y_c) \in \mathbb{R}^2$	Coordonnées du centre de la cellule c.
$(x_p, y_p, a_p) \in \mathbb{R}^3$	Coordonnées de la pose discrète p.
$\mathrm{O} : \mathbb{N} \to \{0, 1\}$	Carte d'**occupation**
$\mathrm{N} : \mathbb{N} \to \{0, 1\}$	Carte de **navigabilité**
$\mathcal{O}$, $\mathcal{O}^C$	Ensembles des cellules occupées, non-occupées
$\mathcal{N}$, $\mathcal{N}^C$	Ensembles des cellules navigables, non-navigables
$p_r, p_r(t)$	Pose du robot à l'instant t
$c_r, c_r(t)$	Cellule du robot à l'instant t
$\mathrm{T} : \mathbb{N}^2 \to \mathbb{R}^+$	**Temps de parcours** entre deux poses
$\mathrm{V} : \mathbb{N}^2 \to \{0, 1\}$	**Visibilité** d'une cellule depuis une autre
$n_i \in \mathbb{N}$	Nombre d'**objets** identifiables
$\mathbf{o}_n \in \mathbb{N}$	Identifiant de la cellule du n-ième objet
$\mathbf{o}_0 \in \mathbb{N}$	Identifiant de la cellule initiale du robot
$\mathrm{D} : \mathbb{N}^2 \to \mathbb{R}$	**Distance** entre deux cellules
$\mathrm{D}_c : (\mathcal{O}^C)^2 \to \mathbb{R}$	Distance de **contournement** entre deux cellules
$\mathrm{A} : \mathbb{N}^2 \to \mathbb{R}$	**Direction** d'une cellule depuis une autre
$\mathbf{i}_n = (\mathbf{d}_n, \mathbf{a}_n)$	**Instruction** pour le n-ième objet
$\mathbf{d}_n \in \mathbb{R}$	Indication sur la distance du n-ième objet
$\mathbf{a}_n \in \mathbb{R}$	Indication sur la direction du n-ième objet
$\mathbf{z} \in \{0, 1\}^{n_i \times n_g}$	**Résultat d'analyse** d'une photo
$\mathbf{z}_n \in \{0, 1\}^{n_g}$	Résultat d'analyse pour le n-ième objet
$\mathbf{z}_{n,c} \in \{0, 1\}$	Observation de la cellule c pour le n-ième objet
$\mathcal{D}_{p_r}$	**Champ de détection** à la pose p_r
$\mathcal{V}_{p_r}$	**Champ de vision** à la pose p_r
$\mathbf{u}(t)$	**Utilité des observations** à l'instant t
$\mathbf{b}(t)$	**Distribution de probabilité** sur les objets

Par la suite, nous notons $\mathcal{O}$ et $\mathcal{N}$ respectivement les ensembles de cellules occupées et navigables. A contrario, $\mathcal{O}^C$ et $\mathcal{N}^C$ représentent respectivement les ensembles de cellules non-occupées et non-navigables (conformément aux notations définies page x).

Déplacements élémentaires

Nous notons $p_r(t)$ la pose discrète du robot à l'instant t et $c_r(t)$ la cellule correspondante. De l'instant t à l'instant t+1, le robot change de pose par déplacement élémentaire : **tourner** d'un huitième de tour vers la gauche ou la droite, **avancer** dans la cellule adjacente à laquelle il fait face. Par extension, une pose discrète est dite *adjacente* à la pose du robot s'il peut s'y rendre en un seul déplacement élémentaire.

Nous définissons $\mathrm{T} : \mathbb{N}^2 \to \mathbb{R}^+$ le **temps de parcours**, qui est une estimation du temps minimum requis pour que le robot se rende d'une pose discrète à une autre par une succession de déplacements élémentaires. Ce temps de parcours est obtenu à l'aide d'une méthode classique de plus court chemin (Dijkstra, 1959) dans le graphe d'adjacence des poses discrètes navigables.

Visibilité d'une cellule

À partir de la carte d'occupation, nous définissons la **fonction de visibilité** $\mathrm{V} : \mathbb{N}^2 \to \{0, 1\}$: une cellule c_2 est visible depuis une cellule c_1, et inversement, si aucune des cellules entre les deux extrémités n'est occupée (figure 2.3, page 18) :

$$\mathrm{V}(c_1, c_2) = \left[\mathrm{E}(c_1, c_2) \cap \mathcal{O} = \emptyset \right] \tag{2.1}$$

avec $\mathrm{E} : \mathbb{N}^2 \to \{0, 1\}$ l'ensemble des cellules intersectées par le segment de droite entre les centres de deux cellules. Même si elle peut être obtenue de manière exacte, cette fonction E peut être approximée, pour de meilleures performances, à l'aide d'une méthode de traçage de lignes discrètes comme, par exemple, l'algorithme de Bresenham (1965).

2.2 Objets et instructions

Une série de $n_i \in \mathbb{N}$ objets identifiables et uniques sont placés dans l'environnement. On définit le vecteur des cellules des objets $\mathbf{o} \in \mathbb{N}^{n_i}$. Chaque composante $\mathbf{o}_n \in \mathbb{N}$ de ce vecteur est l'identifiant de la cellule contenant le n-ième objet ($n = 1 \ldots n_i$). Par abus de notation, nous notons $\mathbf{o}_0$ l'identifiant de la cellule initiale du robot.

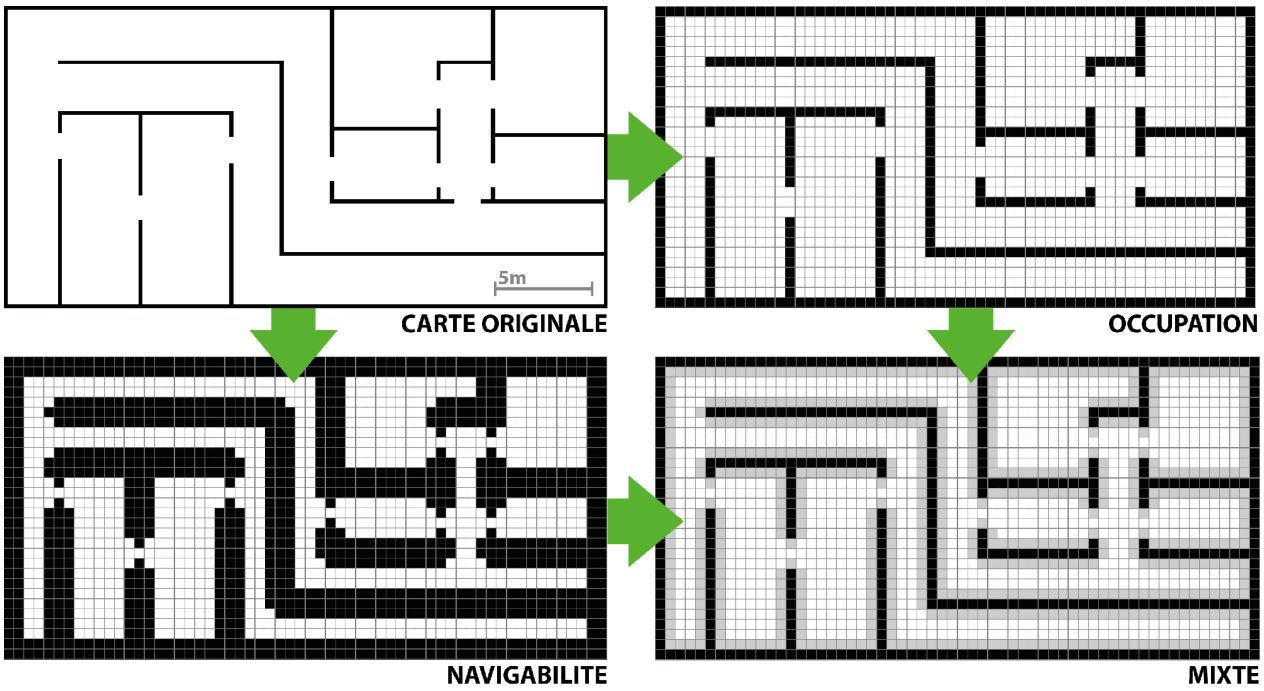


Figure 2.1 **Cartes d'occupation et de navigabilité**. Étant donné une carte de l'environnement, on détermine l'occupation des cellules de la grille (cellules noires occupées). En tenant compte des dimensions du robot, on détermine les cellules navigables (cellules blanches navigables).

2.2.1 Indications de distance et direction

Chaque instruction indique approximativement la position du n-ième objet par rapport à l'objet précédent en donnant deux indications : la **distance** $\mathbf{d}_n$ et la **direction** $\mathbf{a}_n$ (figure 2.2). En notant $(x_c, y_c) \in \mathbb{R}^2$ les coordonnées du centre d'une cellule c, définissons la distance $\mathrm{D} : \mathbb{N}^2 \to \mathbb{R}$ et la direction $\mathrm{A} : \mathbb{N}^2 \to \mathbb{R}$:

$$\mathrm{D}(c_1, c_2) = \sqrt{(x_{c_2} - x_{c_1})^2 + (y_{c_2} - y_{c_1})^2} \tag{2.2}$$

$$\mathrm{A}(c_1, c_2) = \mathrm{atan2}\Big(y_{c_2} - y_{c_1},\, x_{c_2} - x_{c_1} \Big) \tag{2.3}$$

Les indications $\mathbf{a}_n$ sur la direction et $\mathbf{d}_n$ sur la distance sont approximatives et diffèrent de leurs valeurs exactes $\mathrm{D}(\mathbf{o}_{n-1}, \mathbf{o}_n)$ et $\mathrm{A}(\mathbf{o}_{n-1}, \mathbf{o}_n)$. Nous les modélisons par deux distributions

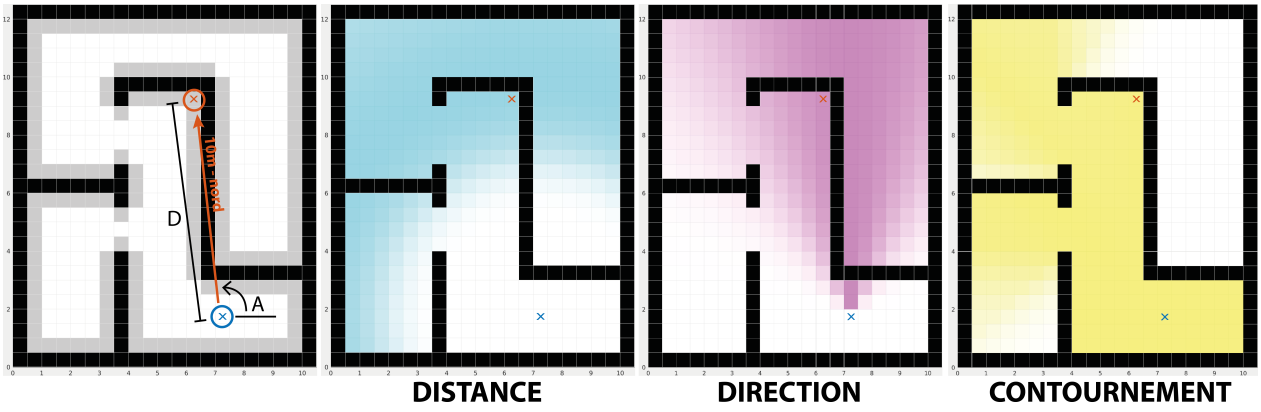


Figure 2.2 **Modèle d'instruction**. Probabilité des indications de distance $\mathbf{d}_n$ (cyan) et de direction $\mathbf{a}_n$ (magenta) en fonction de la cellule $\mathbf{X}_n$ de l'objet associé, étant donné la cellule $\mathbf{x}_{n-1}$ de l'objet précédent (équations 2.4 et 2.5). Distribution sur la cellule $\mathbf{X}_n$ d'un objet étant donné la cellule $\mathbf{x}_{n-1}$ de l'objet précédent d'après le modèle de contournement (jaune, équation 2.7).

normales indépendantes l'une de l'autre et centrées autour des valeurs réelles :

$$\Big(\mathbf{D}_n \,\Big|\, \mathbf{o}_n,\, \mathbf{o}_{n-1} \Big) \sim \mathcal{N}\Big(\mathrm{D}(\mathbf{o}_{n-1}, \mathbf{o}_n),\, \big(\mathrm{D}(\mathbf{o}_{n-1}, \mathbf{o}_n) \cdot \sigma_{i,d}\big)^2 \Big) \tag{2.4}$$

$$\Big(\mathbf{A}_n \,\Big|\, \mathbf{o}_n,\, \mathbf{o}_{n-1} \Big) \sim \mathcal{N}\Big(\mathrm{A}(\mathbf{o}_{n-1}, \mathbf{o}_n),\, \sigma_{i,a}^2 \Big) \tag{2.5}$$

avec les paramètres $\sigma_{i,a}$ et $\sigma_{i,d}$ pouvant être évalués expérimentalement. Notons que considérer un écart-type sur l'indication en distance proportionnel à la distance réelle équivaut à considérer une distribution normale sur l'**erreur relative** en distance. Nous modélisons ainsi le fait qu'il est d'autant plus aisé d'estimer une distance que celle-ci est courte.

Pour simplifier les notations, nous notons $\mathbf{i}_n = (\mathbf{a}_n, \mathbf{d}_n)$ la paire des indications sur la direction et la distance. Ainsi, avec l'indépendance statistique des indications $\mathbf{a}_n$ et $\mathbf{d}_n$, la **probabilité de la n-ième instruction** étant donné les cellules des objets associés est notée :

$$\begin{aligned} P\Big(\mathbf{I}_n \,\Big|\, \mathbf{o}_n,\, \mathbf{o}_{n-1} \Big) &= P\Big(\mathbf{A}_n,\, \mathbf{D}_n \,\Big|\, \mathbf{o}_n,\, \mathbf{o}_{n-1} \Big) \\ &= P\Big(\mathbf{A}_n \Big|\, \mathbf{o}_n,\, \mathbf{o}_{n-1} \Big) \cdot P\Big(\mathbf{D}_n \Big|\, \mathbf{o}_n,\, \mathbf{o}_{n-1} \Big) \end{aligned} \tag{2.6}$$

Les probabilités des indications $\mathbf{a}_n$ sur la direction et $\mathbf{d}_n$ sur la distance étant donné les cellules des objets associés sont illustrées à la figure 2.2.

2.2.2 Distance de contournement

Si on donne au robot une instruction spécifiant qu'un objet se trouve, par exemple, *à dix mètres sur sa droite*, on sous-entend par hypothèse qu'il peut l'atteindre directement, en parcourant une trajectoire globalement rectiligne. Si, *dix mètres sur sa droite*, il y a effectivement une cellule non-occupée, le robot devrait pouvoir s'y rendre en parcourant *environ dix mètres*. S'il doit faire le tour du bâtiment pour y parvenir, ce n'est certainement pas l'endroit auquel le guide pensait.

Nous définissons la **distance de contournement** $\mathrm{D}_c : (\mathcal{O}^C)^2 \to \mathbb{R}$ comme la distance minimale entre deux cellules non-occupées en contournant toutes les cellules occupées. Cette distance est obtenue à l'aide d'une méthode classique de plus court chemin (Dijkstra, 1959) dans le graphe d'adjacence des cellules non-occupées de la grille.

Dès lors, pour qu'une instruction ait du sens, la distance de contournement $\mathrm{D}_c(\mathbf{o}_n, \mathbf{o}_{n-1})$ entre deux objets doit rester proche de la distance euclidienne $\mathrm{D}(\mathbf{o}_n, \mathbf{o}_{n-1})$. Nous proposons une distribution de probabilité conditionnelle sur la cellule $\mathbf{o}_n$ d'un objet étant donné la cellule $\mathbf{o}_{n-1}$ de l'objet précédent diminuant avec la différence entre ces deux distances :

$$P\Big(\,\mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1}\,\Big) \propto e^{-\frac{1}{\sigma_{i,r}}\left(\mathrm{D}_c(\mathbf{o}_n,\mathbf{o}_{n-1})-\mathrm{D}(\mathbf{o}_n,\mathbf{o}_{n-1})\right)^2} \tag{2.7}$$

avec un paramètre $\sigma_{i,r}$, représentatif de la déviation attendue entre la distance réelle et la distance de contournement, à évaluer expérimentalement. Cette distribution est normalisée sur l'ensemble des cellules non-occupées de l'environnement (figure 2.2).

2.3 Détecteur d'objet

À chaque pose discrète navigable de l'environnement, le robot peut prendre une photo et l'analyser avec un algorithme de détection d'objet. Typiquement, un algorithme de ce type analyse une photo à la recherche d'objets d'une classe donnée et retourne un ensemble de détections. Chaque détection spécifie la position de l'objet en donnant une estimation de sa distance et de sa direction relatives au robot.

2.3.1 Discrétisation du résultat d'analyse

Pour simplifier le modèle, nous discrétisons le résultat de l'analyse d'une photo par l'algorithme de détection. Plutôt que de considérer une position approximative indiquée par une

distance et un angle, nous discrétisons les détections sur la grille en considérant les cellules dans lesquelles l'objet a été détecté. Les coordonnées $(x, y) \in \mathbb{R}^2$ associées à une détection à une distance d et dans une direction a sont :

$$\begin{aligned} x &= x_{p_r} + d \cdot \cos(a + a_{p_r}) \\ y &= y_{p_r} + d \cdot \sin(a + a_{p_r}) \end{aligned} \tag{2.8}$$

avec $(x_{p_r}, y_{p_r}) \in \mathbb{R}^2$ les coordonnées du robot et $a_{p_r} \in \mathbb{R}$ son orientation. Nous notons alors qu'une cellule *contient la détection* si elle contient ce point de coordonnées (x, y).

Plutôt que de représenter le résultat de l'analyse d'une photo par l'algorithme comme un ensemble de détections, nous le représentons sous forme d'une matrice booléenne creuse $\mathbf{z}$. Chaque élément $\mathbf{z}_{n,c}$ de cette matrice indique si la cellule c contient au moins une des détections du résultat de l'analyse pour l'objet n :

$$\Big(\mathbf{z}_{n,c} = 1\Big) \quad \triangleq \quad \begin{array}{l} \textit{Dans le résultat de l'analyse de la photo} \\ \textit{pour l'objet } n\textit{, la cellule } c \textit{ contient au} \\ \textit{moins une détection.} \end{array}$$

Pour plus de clarté dans les sections suivantes, nous nommons la matrice $\mathbf{z}$ le *résultat d'analyse* d'une photo, le vecteur $\mathbf{z}_n$ le *résultat d'analyse pour l'objet n* et l'élément $\mathbf{z}_{n,c}$ l'*observation pour l'objet n de la cellule c*. Par analogie, nous pouvons en effet imaginer que le détecteur d'objet *observe* successivement chaque cellule dans son champ de vision pour déterminer si l'objet s'y trouve. Nous commençons par définir un modèle d'observation $\mathbf{z}_{n,c}$ d'une cellule unique et nous nous en servons ensuite pour modéliser les résultats d'analyse $\mathbf{z}_n$ pour chaque objet et, par extension, $\mathbf{z}$ de la photo complète.

2.3.2 Champ de détection et champ de vision

L'ensemble des coordonnées (x, y) pouvant être associées à une détection (équation 2.8) est limité par les caractéristiques de la caméra utilisée. Ainsi, la direction a d'une détection sera limitée par l'angle de champ α_d de la caméra, et la distance d par des distances minimale $\rho_{d,l}$ et maximale $\rho_{d,h}$ de détection :

$$a \in \left[-\frac{\alpha_d}{2}, \frac{\alpha_d}{2} \right] \qquad\qquad d \in \left[\rho_{d,l}, \rho_{d,h} \right]$$

Dans cette optique, nous définissons le **champ de détection** $\mathcal{D}_{p_r}$ du robot à la pose p_r comme l'ensemble des cellules de l'environnement pouvant contenir une détection d'après

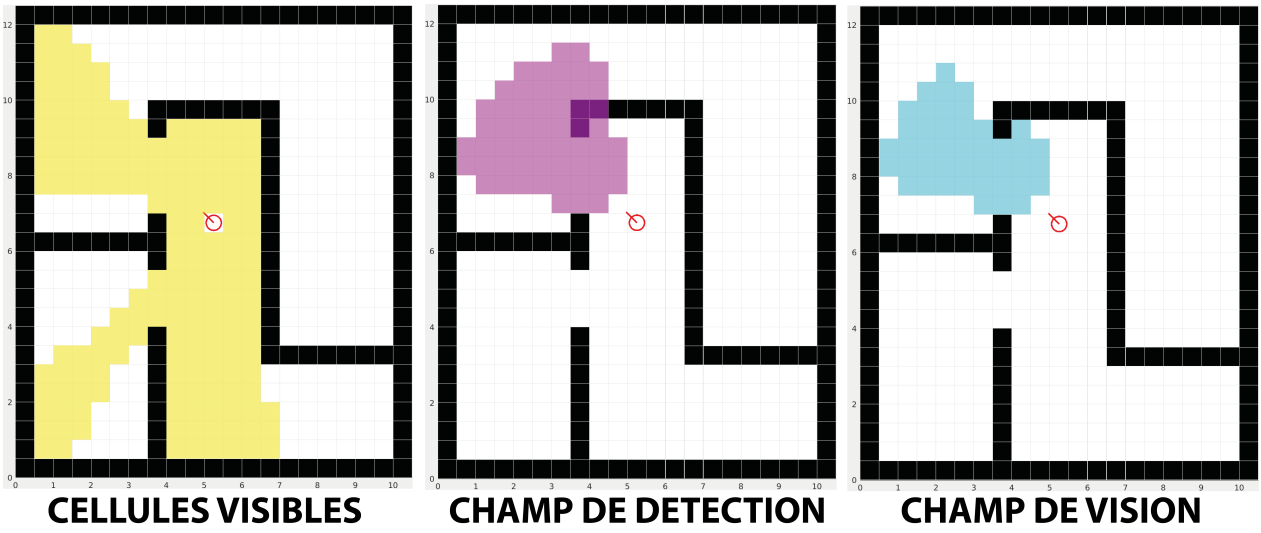


Figure 2.3 **Cellules visibles, champ de vision et champ de détection**. Le champ de détection est l'ensemble des cellules pouvant contenir une détection d'après les limites physiques du détecteur. Le champ de vision est l'ensemble des cellules du champ de détection qui sont visibles d'après la carte d'occupation.

les contraintes physiques de la caméra (figure 2.3). Par souci de performance, nous pouvons approximer ce champ de détection comme l'ensemble des cellules dont le centre $(x_c, y_c) \in \mathbb{R}^2$ pourrait être associé à une détection :

$$\mathcal{D}_{p_r} \triangleq \left\{ c \in \mathbb{N} \,\middle|\, \mathrm{D}(c_r, c) \in \left[\, \rho_{d,l},\, \rho_{d,h} \,\right] \wedge \left| a_{p_r} - \mathrm{A}(c_r, c) \right| \in \left[\, -\frac{\alpha_d}{2},\, \frac{\alpha_d}{2} \,\right] \right\}$$

avec c_r la cellule du robot, $\mathrm{D}(c_r, c)$ la distance entre les cellules c_r et c (équation 2.2) et $\mathrm{A}(c_r, c)$ la direction de la cellule c depuis la cellule c_r (équation 2.3).

Par extension, nous définissons le **champ de vision** $\mathcal{V}_{p_r} \subset \mathcal{D}_{p_r}$ du robot à la pose p_r comme l'ensemble des cellules du champ de détection qui sont visibles d'après la carte d'occupation. Avec la visibilité V définie précédemment (équation 2.1), nous avons :

$$\mathcal{V}_{p_r} \triangleq \left\{ c \in \mathcal{D}_{p_r} \,\middle|\, \mathrm{V}(c_r, c){=}1 \right\} \tag{2.9}$$

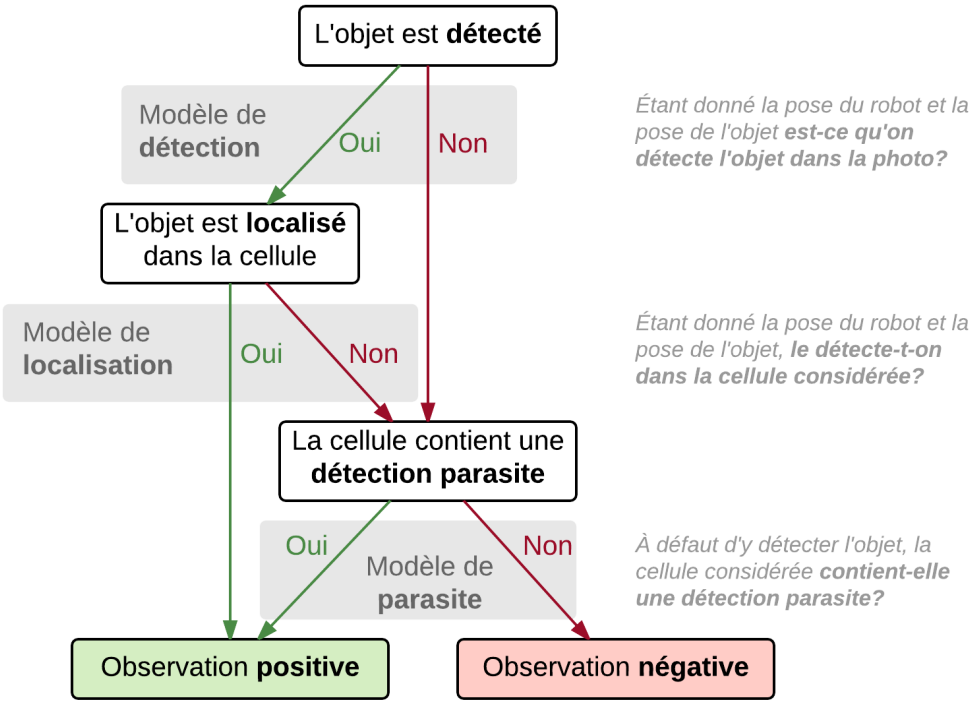


Figure 2.4 **Modèle d'observation d'une cellule**. Illustration des différents cas possibles pour l'obtention d'une observation positive ou négative dans une cellule de l'environnement.

2.3.3 Modèle d'observation d'une cellule

L'algorithme de détection n'est pas parfait. D'une part, une détection peut réellement être due à un objet de la classe considérée (*vrai-positif*), mais peut également être une erreur de l'algorithme de détection : une détection parasite (*faux-positif*). D'autre part, si une détection est effectivement due à un objet, la distance et l'orientation indiquées ne sont que des valeurs approchées de la distance et de l'orientation réelles.

Dans ce contexte, le **modèle d'observation d'une cellule** donne la probabilité d'une observation $\mathbf{z}_{n,c}$ dans une cellule étant donné la pose p_r du robot et la cellule $\mathbf{o}_n$ de l'objet :

$$P\Big(\mathbf{Z}_{n,c} \,\Big|\, \mathbf{o}_n,\, p_r\Big)$$

Comme illustré à la figure 2.4, nous définissons ce modèle en considérant les différentes sources d'erreurs dans trois sous-modèles :

1. Un **modèle de détection** donne la probabilité de détecter l'objet (*vrai-positif*) étant donné sa cellule $\mathbf{o}_n$ et la pose p_r du robot. De manière similaire à Atanasov et al. (2014b), nous considérons une probabilité de détection diminuant avec la distance pour les cellules du champ de vision. Ainsi, en définissant l'événement $\mathbf{D}_n \triangleq$ *« Détecter correctement l'objet n dans la photo »*, nous avons :

$$P\Big(\mathbf{D}_n \,\Big|\, \mathbf{o}_n,\, p_r\Big) = \begin{cases} \pi_{d,n} \cdot e^{-\left(\frac{\mathrm{D}(c_r, \mathbf{o}_n)}{\sigma_{d,n}}\right)} & \text{si } \mathbf{o}_n \in \mathcal{V}_{p_r} \\ 0 & \text{sinon} \end{cases} \tag{2.10}$$

avec D la distance euclidienne (équation 2.2) et $\mathcal{V}_{p_r}$ le champ de vision depuis la pose p_r (équation 2.9). Les paramètres $\pi_{d,n}$ et $\sigma_{d,n}$ varient pour différentes classes d'objets et peuvent être déterminés expérimentalement (voir section 4.1, page 63).

2. Un **modèle de localisation** donne la probabilité que l'algorithme de détection localise l'objet détecté dans une cellule donnée étant donné la cellule $\mathbf{o}_n$ le contenant réellement. Nous considérons une probabilité de localisation suivant une distribution normale bidimensionnelle autour du centre de la cellule $\mathbf{o}_n$ de l'objet. En notant $(x_{\mathbf{o}_n}, y_{\mathbf{o}_n}) \in \mathbb{R}$ les coordonnées du centre de la cellule $\mathbf{o}_n$ et $(x, y) \in \mathbb{R}^2$ les coordonnées auxquelles l'objet est localisé par l'algorithme de détection (équation 2.8), nous avons :

$$\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| x_{\mathbf{o}_n},\, y_{\mathbf{o}_n} \right) \sim \mathcal{N}\left(\begin{bmatrix} x_{\mathbf{o}_n} \\ y_{\mathbf{o}_n} \end{bmatrix}, \begin{bmatrix} \sigma_l^2 & 0 \\ 0 & \sigma_l^2 \end{bmatrix} \right) \tag{2.11}$$

avec σ_l un écart-type à déterminer expérimentalement. Cette distribution est ensuite discrétisée par intégration afin d'obtenir la probabilité de localisation dans chacune des cellules avoisinantes. Définissons l'événement $\mathbf{L}_{n,c} \triangleq$ *« Localiser dans la cellule c l'objet n correctement détecté »* pour écrire :

$$P\Big(\mathbf{L}_{n,c} \,\Big|\, \mathbf{o}_n\Big) = \iint\limits_{(x,y)\in c} P\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| x_{\mathbf{o}_n},\, y_{\mathbf{o}_n} \right) dx\, dy \tag{2.12}$$

Notons que ce modèle de localisation est assez simpliste, négligeant notamment une influence certaine de la distance à l'objet dans l'amplitude de l'erreur de localisation, ou encore des erreurs distinctes sur la distance et l'orientation indiquées. Cependant, nous supposons par hypothèse des petites erreurs de localisation, de l'ordre d'une cellule, et un modèle plus complexe n'est dès lors pas indispensable.

3. Un **modèle de parasite** donne la probabilité pour une cellule donnée de contenir une détection parasite. Pour la modéliser, nous pouvons imaginer que la cause d'une détection parasite dans une cellule est la présence dans celle-ci d'éléments de l'environnement pouvant être mépris pour un objet de la classe considérée. Nous considérons ainsi une probabilité de détection parasite constante pour toutes les cellules du champ de détection à l'exception de la cellule contenant l'objet. Définissons l'événement $\mathbf{P}_{n,c} \triangleq$ « *La cellule c contient une détection parasite de l'objet n* » pour écrire :

$$P\Big(\mathbf{P}_{n,c}\,\Big|\,\mathbf{o}_n\Big) = \begin{cases} \pi_{p,n} & \text{si } c \in \mathcal{D}_{p_r} \text{ et } c \neq \mathbf{o}_n \\ 0 & \text{sinon} \end{cases} \tag{2.13}$$

avec $\pi_{p,n}$ un paramètre à déterminer expérimentalement. La raison pour laquelle la probabilité de détection parasite est nulle dans la cellule $\mathbf{o}_n$ contenant l'objet est que, si on détecte un objet dans cette cellule, cette détection est correcte par définition.

D'après l'illustration du modèle d'observation à la figure 2.4, l'observation $\mathbf{z}_{n,c}$ d'une cellule c à la recherche du n-ième objet peut être positive dans deux cas :

1. La cellule c contient une détection parasite (*faux-positif*).
2. La cellule c ne contient pas de détection parasite, mais :
   (a) L'objet est détecté dans le champ de vision (*vrai-positif*),
   (b) Et cette détection est localisée dans la cellule c.

Avec les sous-modèles de détection, de localisation et de parasite définis ci-dessus, nous formulons sur la base de ces cas la **probabilité d'observation d'une cellule** c étant donné la pose p_r du robot et la cellule $\mathbf{o}_n$ contenant l'objet :

Modèle d'observation d'une cellule

$$P\Big(\mathbf{Z}_{n,c}\Big) = \underbrace{P\Big(\mathbf{D}_n\Big) \cdot P\Big(\mathbf{L}_{n,c}\Big)}_{\substack{\text{Détecté et localisé} \\ \text{dans la cellule } c}} \cdot \Big(1 - P\Big(\mathbf{P}_{n,c}\Big)\Big) + \underbrace{P\Big(\mathbf{P}_{n,c}\Big)}_{\substack{\text{Détection} \\ \text{parasite}}} \tag{2.14}$$

où, pour clarifier les notations, nous avons omis d'indiquer les dépendances conditionnelles en $\mathbf{o}_n$ et p_r. Observons qu'en l'absence d'erreur de localisation ($\sigma_l = 0$, équation 2.11), le

modèle d'observation d'une cellule devient :

$$P\Big(\mathbf{Z}_{n,c}\,\Big|\,p_r,\,\mathbf{o}_n\Big) = \begin{cases} P\Big(\mathbf{D}_n\,\Big|\,p_r,\,\mathbf{o}_n\Big)\cdot\Big(1-P\Big(\mathbf{P}_{n,c}\,\Big|\,\mathbf{o}_n\Big)\Big) & \text{si } c=\mathbf{o}_n \\ P\Big(\mathbf{P}_{n,c}\,\Big|\,\mathbf{o}_n\Big) & \text{sinon} \end{cases}$$

En tenant compte de la probabilité de détection parasite nulle pour la cellule contenant l'objet (équation 2.13), nous obtenons finalement un **modèle d'observation simplifié** :

$$P\Big(\mathbf{Z}_{n,c}\,\Big|\,p_r,\,\mathbf{o}_n\Big) = \begin{cases} P\Big(\mathbf{D}_n\,\Big|\,p_r,\,\mathbf{o}_n\Big) & \text{si } c=\mathbf{o}_n \\ \pi_{p,n} & \text{sinon} \end{cases} \tag{2.15}$$

Dans ce cas, l'observation d'une cellule donnée est indépendante du contenu des autres cellules et ne dépend, pour la cellule contenant l'objet, que de la probabilité de détection et, pour les autres, de la probabilité de détection parasite.

2.3.4 Modèle de résultat d'analyse d'une photo

En faisant l'hypothèse d'indépendance conditionnelle des observations $\mathbf{z}_{n,c}$ de chaque cellule, la probabilité de l'analyse $\mathbf{z}_n$ d'une photo complète pour l'objet n est simplement donnée par le produit des probabilités des observations $\mathbf{z}_{n,c}$ pour chaque cellule du champ de détection.

$$P\Big(\mathbf{z}_n\,\Big|\,p_r,\,\mathbf{o}_n\Big) = \prod_{c\in\mathcal{D}_{pr}} P\Big(\mathbf{z}_{n,c}\,\Big|\,p_r,\,\mathbf{o}_n\Big) \tag{2.16}$$

où nous avons tenu compte du fait que l'observation d'une cellule en dehors du champ de détection sera toujours négative. Si on considère que le détecteur d'objet localise parfaitement les objets, cette hypothèse d'indépendance conditionnelle des résultats est justifiée. Comme on l'a vu plus haut (équation 2.15), la probabilité d'observation d'une cellule dépend dans ce cas exclusivement du contenu de cette cellule : que cela corresponde à un objet identifiable ou à d'autres éléments susceptibles de causer une détection parasite. Maintenant, avec une localisation imparfaite, ce n'est plus le cas : le résultat d'analyse d'une cellule dépend du contenu des cellules environnantes. Dans la mesure ou nous considérons une erreur de localisation faible, nous supposons que ce modèle reste toutefois acceptable.

Finalement, la probabilité du résultat d'analyse $\mathbf{z}$ correspond à la probabilité jointe des résultats d'analyse $\mathbf{z}_n$ pour chaque objet :

$$P\Big(\,\mathbf{z}\,\Big|\,p_r,\,\mathbf{o}\,\Big) = P\Big(\,\mathbf{z}_1 \ldots \mathbf{z}_{n_i}\,\Big|\,p_r,\,\mathbf{o}\,\Big)$$

Nous faisons l'hypothèse d'indépendance des résultats d'analyse $\mathbf{z}_n$ pour chaque objet : la confusion d'un objet pour un autre est considérée comme une détection parasite. Ainsi, la probabilité du résultat d'analyse $\mathbf{z}$ est simplement donnée par le produit des résultats d'analyse $\mathbf{z}_n$ pour chaque objet :

$$P\Big(\,\mathbf{z}\,\Big|\,p_r,\,\mathbf{o}\,\Big) = \prod_{n=1}^{n_i} P\Big(\,\mathbf{z}_n\,\Big|\,p_r,\,\mathbf{o}_n\Big) \tag{2.17}$$

2.3.5 Dépendance des résultats d'analyse

Jusqu'à maintenant, nous avons seulement considéré le résultat d'analyse $\mathbf{z}$ d'une photo unique. Cependant, en se déplaçant dans l'environnement, le robot va prendre et analyser une successions de photos. Étant donné les cellules contenant les objets, les résultats d'analyse de ces photos successives ne sont pas statistiquement indépendants.

Notons $\mathbf{z}(t)$ le résultat de l'analyse d'une photo à partir de la pose $p_r(t)$ du robot à l'instant t. Imaginons qu'à deux instants distincts, le robot prenne une photo depuis la même pose $p_r(t_1) = p_r(t_2)$. Nous faisons l'hypothèse d'un environnement statique et négligeons le bruit introduit par la caméra, d'éventuelles variations de luminosité ou obstructions temporaires. Les photos à ces deux instants seront donc identiques, et les résultats d'analyse $\mathbf{z}(t_1)$ et $\mathbf{z}(t_2)$ également. Ces deux résultats d'analyse ne sont donc pas conditionnellement indépendants étant donné les cellules $\mathbf{o}$ des objets et les poses $p_r(t)$ du robot :

$$P\Big(\,\mathbf{Z}(t_2){=}\mathbf{z}(t_1)\,\Big|\,\mathbf{z}(t_1),\,\mathbf{o},\,p_r(t_1),\,p_r(t_2)\,\Big) = 1 \neq P\Big(\,\mathbf{Z}(t_2){=}\mathbf{z}(t_1)\,\Big|\,\mathbf{o},\,p_r(t_2)\,\Big)$$

De manière similaire, comme illustré à la figure 2.5, le robot peut tourner sur lui même de telle sorte que des sections de deux photos successives coïncident. Les résultats d'analyse $\mathbf{z}(t_1)$ et $\mathbf{z}(t_2)$ pour ces sections coïncidentes seront probablement identiques et, de nouveau, ils ne sont pas indépendants.

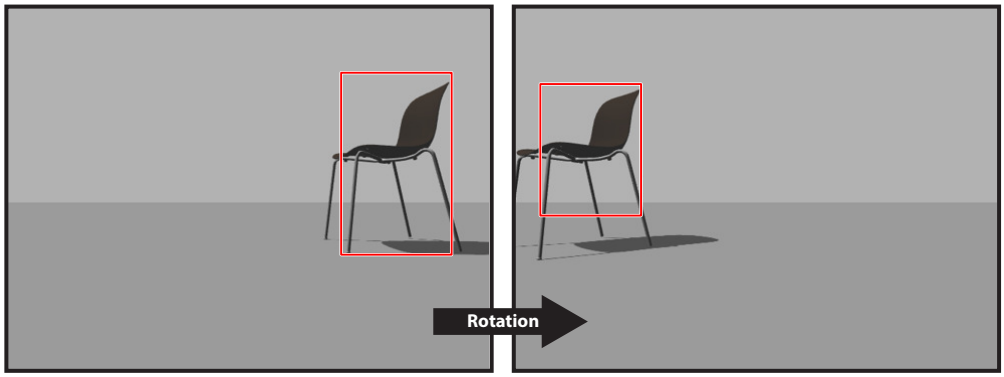


Figure 2.5 **Dépendance des résultats d'analyse**. Le robot tourne sur le lui même et la chaise reste dans son champ de vision. S'il la détecte dans la première photo, il y a de fortes chances qu'il la détecte également dans la seconde : les résultats ne sont pas indépendants.

Intuitivement, cette dépendance conditionnelle entre résultats d'analyse est d'autant plus importante que les poses correspondantes sont semblables. À défaut d'être capable de modéliser précisément cette dépendance, nous formulons l'hypothèse suivante :

Deux **observations** $\mathbf{z}_{n,c}(t_1)$ et $\mathbf{z}_{n,c}(t_2)$ d'une même cellule c, prise en photo et analysée depuis les poses $p_r(t_1)$ et $p_r(t_2)$, sont **identiques** si, simultanément :

1. La cellule c appartient aux deux champs de détection $\mathcal{D}_{p_r(t_1)}$ et $\mathcal{D}_{p_r(t_2)}$.
2. Les photos sont prises depuis une même cellule $c_r(t_1) = c_r(t_2)$.

Dans tous les autres cas, les observations $\mathbf{z}_{n,c}(t_1)$ et $\mathbf{z}_{n,c}(t_2)$ sont **conditionnellement indépendantes** étant donné la cellule $\mathbf{o}_n$ contenant l'objet.

Autrement dit, tant que le robot reste à la même position, même s'il tourne sur lui-même (figure 2.5), on considère que l'algorithme de détection retournera toujours les mêmes détections (détections parasites comprises) pour des sections correspondantes de son champ de détection. Nous formulons cette hypothèse mathématiquement :

Hypothèse de dépendance des observations

$$P\Big(\,\mathbf{Z}_{n,c}(t_2){=}\mathbf{z}_{n,c}(t_1)\,\Big|\,\mathbf{z}_{n,c}(t_1),\,\mathbf{o}_n,\,p_r(t_1),\,p_r(t_2)\,\Big)$$

$$= \begin{cases} 1 \text{ si } \begin{cases} c_r(t_1) = c_r(t_2) & \text{Depuis la même cellule.} \\ c \in \Big(\mathcal{D}_{p_r(t_1)} \cap \mathcal{D}_{p_r(t_2)}\Big) & \text{Dans les deux champs de détection.} \end{cases} \\ P\Big(\,\mathbf{Z}_{n,c}(t_2){=}\mathbf{z}_{n,c}(t_1)\,\Big|\,\mathbf{o}_n,\,p_r(t_2)\,\Big) \text{ sinon} \end{cases} \tag{2.18}$$

2.3.6 Utilité des observations

Nous définissons l'**utilité des observations** $\mathbf{u}(t) \in \{0,1\}^{n_c \times n_c}$: une matrice booléenne permettant de conserver un historique des observations réalisées préalablement qui, d'après l'hypothèse de dépendance 2.18, resteront identiques dans les analyses futures. Ainsi, chaque composante $\mathbf{u}_{c_r,c}(t)$ de cette matrice d'utilité indique si la cellule c a déjà été fait partie du champ de détection à un instant $t' < t$ où le robot se trouvait dans la cellule c_r :

$$\Big(\,\mathbf{u}_{c_r,c}(t) = 1\,\Big) \quad \triangleq \quad \left(\nexists\, t' < t \,\middle|\, \Big[\, c_r(t') = c_r \,\Big] \,\wedge\, \Big[\, c \in \mathcal{D}_{p_r(t')} \Big) \,\Big] \right)$$

En pratique, comme illustré à la figure 2.6, l'utilité des observations est mise à jour après chaque analyse en spécifiant comme inutiles l'ensemble des cellule faisant parties du champ de détection $\mathcal{D}_{p_r(t)}$ depuis la cellule $c_r(t)$:

$$\mathbf{u}_{c_r,c}(t{+}1) = \begin{cases} 0 & \text{si } c \in \mathcal{D}_{p_r(t)} \text{ et } c_r = c_r(t) \\ \mathbf{u}_{c_r,c}(t) & \text{sinon} \end{cases} \tag{2.19}$$

Notons qu'un résultat d'analyse $\mathbf{z}(t)$ respectant l'hypothèse de dépendance 2.18 est conditionnellement indépendant des résultats d'analyse précédents étant donné cette matrice d'utilité $\mathbf{u}(t)$, les cellules $\mathbf{o}_n$ des objets et la pose $p_r(t)$ du robot :

$$P\Big(\,\mathbf{z}(t)\,\Big|\,\mathbf{o},\,p_r(0)\ldots p_r(t),\,\mathbf{z}(0)\ldots\mathbf{z}(t{-}1),\,\mathbf{u}(t)\,\Big) = P\Big(\,\mathbf{z}(t)\,\Big|\,\mathbf{o},\,p_r(t),\,\mathbf{u}(t)\,\Big)$$

Dans ce résultat d'analyse $\mathbf{z}(t)$, seules les observations $\mathbf{z}_{n,c}(t)$ des cellules utiles et dans le champ de détection sont préalablement inconnues. Les autres observations ont déjà été obtenues précédemment (observations inutiles) ou sont extérieures au champ de détection (négatives par hypothèse). Nous adaptons donc l'équation 2.16 pour y inclure la dépendance aux résultats d'analyse précédents et finalement définir la **probabilité du résultat d'ana-**

lyse d'une photo pour l'objet n étant donné la cellule $\mathbf{o}_n$ de l'objet à identifier et la pose $p_r(t)$ du robot :

Modèle de résultat d'analyse d'une photo

$$P\Big(\mathbf{z}_n(t)\,\Big|\,p_r(t),\,\mathbf{o}_n,\,\mathbf{u}(t)\Big) = \prod_{c\in\mathcal{U}} P\Big(\mathbf{z}_{n,c}(t)\,\Big|\,p_r(t),\,\mathbf{o}_n\Big) \tag{2.20}$$

$$\text{avec } \mathcal{U} \triangleq \left\{ c \in \mathcal{D}_{p_r(t)} \,\middle|\, \mathbf{U}_{c_r(t),c}(t) \right\}$$

Observation utile et dans le champ de détection.

Les résultats d'analyse $\mathbf{z}_n(t)$ pour chaque objet sont toujours indépendants et la probabilité du résultat d'analyse complet $\mathbf{z}$ est obtenu par multiplication (équation 2.17).

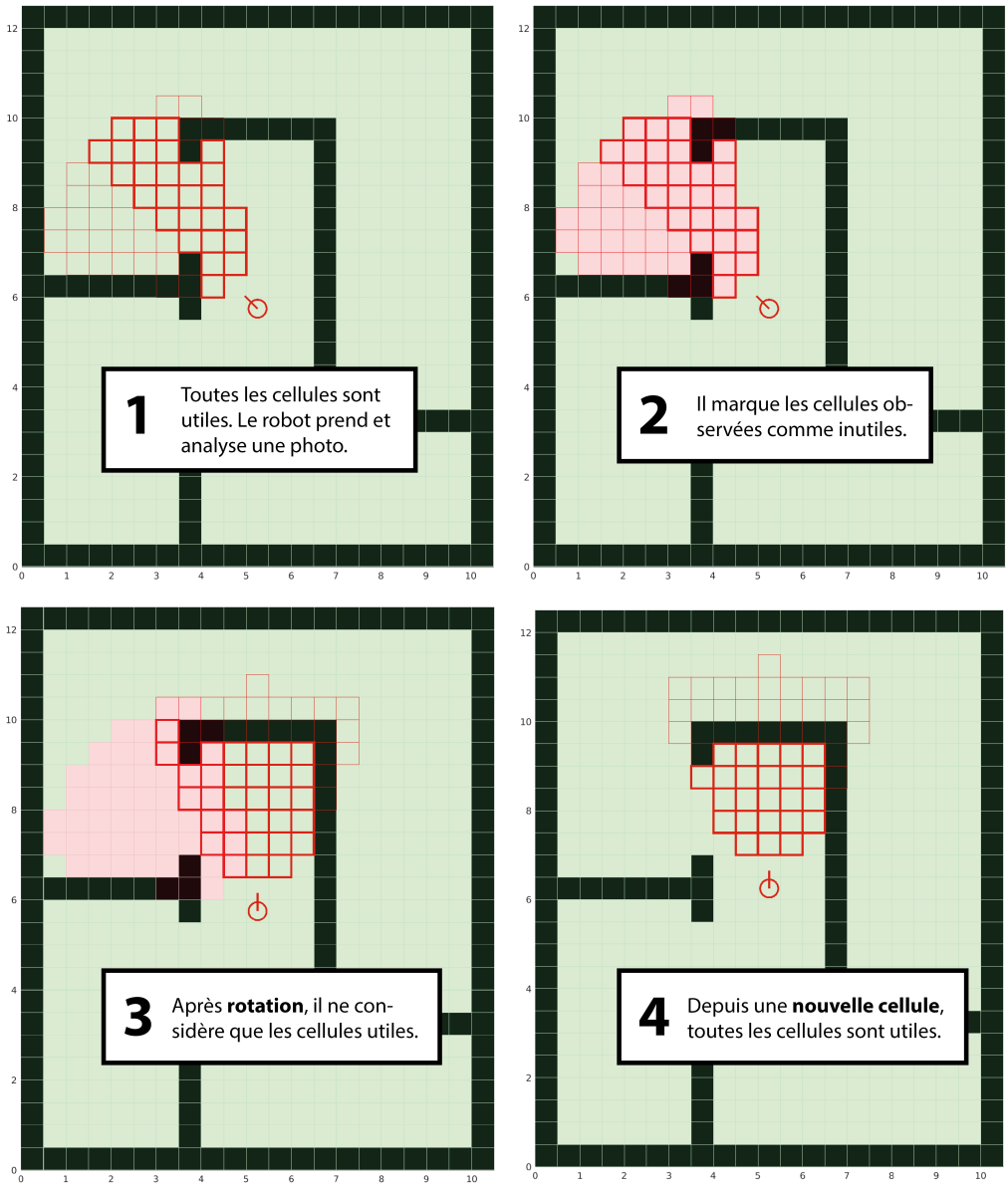


Figure 2.6 **Utilité des observations**. L'observation d'une cellule est utile (en vert) si cette cellule n'a pas encore été dans le champ de détection dans une photo prise depuis la même cellule.

2.4 Processus Décisionnel de Markov

Nous présentons maintenant les éléments constitutifs du processus décisionnel de Markov tel que défini dans l'introduction (section 1.1.3, page 6). Nous commençons par définir l'état du système. Nous présentons ensuite l'espace d'action et justifions les récompenses associées d'après les objectifs poursuivis. Finalement, nous définissons formellement la fonction de transition sur la base des modèles présentés ci-dessus.

2.4.1 État du système

Notons $\mathbf{b}(t)$ la distribution de probabilité jointe sur les cellules $\mathbf{o}$ des objets étant donné l'ensemble des résultats d'analyse obtenus jusqu'à l'instant t compris (le *belief*, par analogie à la théorie des processus décisionnel de Markov partiellement observables) :

$$\mathbf{b}(t) \triangleq P\Big(\mathbf{o} \,\Big|\, \mathbf{z}(1{:}t),\, p_r(1{:}t) \Big)$$

Nous définissons alors l'état $s(t)$ du système comme la pose $p_r(t)$ du robot, cette distribution de probabilité $\mathbf{b}(t)$ et l'utilité des observations $\mathbf{u}(t)$:

$$s(t) \triangleq \Big\{ p_r(t),\, \mathbf{b}(t),\, \mathbf{u}(t) \Big\} \tag{2.21}$$

Par abus de notation, nous notons $\mathbf{b}_n(t)$ la distribution de probabilité marginale sur la cellule du n-ième objet et $\mathbf{b}_{n,c}(t)$ la probabilité que cet objet se trouve dans la cellule c :

$$\mathbf{b}_n(t) \triangleq P\Big(\mathbf{o}_n \,\Big|\, \mathbf{z}(1{:}t),\, p_r(1{:}t) \Big)$$
$$\mathbf{b}_{n,c}(t) \triangleq P\Big(\mathbf{O}_n{=}c \,\Big|\, \mathbf{z}(1{:}t),\, p_r(1{:}t) \Big)$$

2.4.2 Actions et fonction de récompenses

Comme mentionné dans l'introduction, l'objectif du robot est d'*atteindre le dernier objet en évitant de perdre du temps*. Nous définissons ainsi deux types d'actions :

1. **Se déplacer et prendre une photo** : le robot change de pose par *un seul* déplacement élémentaire (*tourner/avancer*), puis prend une photo et l'analyse avec l'algorithme de détection. Nous notons l'action $a(t)$ comme l'identifiant de la pose navigable adjacente

à laquelle le robot se rend :

$$\Big(\, a(t) \in \mathcal{N}_p \,\Big) \triangleq \begin{array}{l} \textit{Tourner ou avancer pour atteindre la} \\ \textit{pose adjacente } a(t)\textit{, puis prendre une} \\ \textit{photo et l'analyser.} \end{array}$$

Nous associons à ce type d'action un coût (récompense négative) prenant en compte la durée du déplacement et le temps nécessaire à l'analyse d'une photo. Avec T_p le temps de parcours (section 2.1, page 11), nous avons :

$$\mathrm{R}\Big(s(t), a(t) \in \mathcal{N}_p\Big) \triangleq -\mathrm{T}_p\Big(p_r(t), a(t)\Big) \tag{2.22}$$

2. **Arrêter de chercher** : le robot pense connaître la position du dernier objet et s'y rend. Il s'agit d'une action terminale et nous notons :

$$\Big(\, a(t) = 0 \,\Big) \triangleq \begin{array}{l} \textit{Arrêter de chercher l'objet et se rendre} \\ \textit{dans la cellule où la probabilité que} \\ \textit{l'objet se trouve est maximale.} \end{array}$$

Nous associons à cette action une récompense positive proportionnelle à la probabilité que l'objet se trouve effectivement dans la cellule sélectionnée, et un coût pour le déplacement jusqu'à cette cellule :

$$\mathrm{R}\Big(s(t), a(t) = 0\Big) \triangleq \kappa \cdot \max_{c \in \mathcal{O}^C} \Big(\, \mathbf{b}_{n_i,c} \,\Big) - \mathrm{T}_p\Big(p_r(t),\, \operatorname*{argmax}_{c \in \mathcal{O}^C} \Big(\, \mathbf{b}_{n_i,c} \,\Big)\Big) \tag{2.23}$$

Intuitivement, le paramètre κ peut être assimilé à la *prudence* du robot. Plus sa valeur est grande, plus la récompense associée à une localisation correcte du dernier objet est importante par rapport au coût du déplacement ; le robot a alors tout intérêt à explorer d'avantage l'environnement pour éviter de faire une erreur. Dans le cas contraire, pour une petite valeur de κ, le robot devrait plutôt minimiser ces déplacements pour arriver rapidement au dernier objet, quitte à faire une erreur sur sa localisation.

2.4.3 Fonction de transition

Avec l'état tel que défini ci-dessus, la fonction de transition se décompose :

$$\begin{aligned}
\mathrm{T}\Big(s(t{+}1), a(t), s(t)\Big) &\triangleq P\Big(\, s(t{+}1) \,\Big|\, a(t),\, s(t) \,\Big) \\
&= P\Big(\, p_r(t{+}1),\, \mathbf{b}(t{+}1),\, \mathbf{u}(t{+}1) \,\Big|\, a(t),\, p_r(t),\, \mathbf{b}(t),\, \mathbf{u}(t) \,\Big) \\
&= P\Big(\, p_r(t{+}1) \,\Big|\, a(t),\, p_r(t),\, \mathbf{b}(t),\, \mathbf{u}(t) \,\Big) \\
&\quad \cdot P\Big(\, \mathbf{u}(t{+}1) \,\Big|\, p_r(t{+}1),\, a(t),\, p_r(t),\, \mathbf{b}(t),\, \mathbf{u}(t) \,\Big) \\
&\quad \cdot P\Big(\, \mathbf{b}(t{+}1) \,\Big|\, \mathbf{u}(t{+}1),\, p_r(t{+}1),\, a(t),\, p_r(t),\, \mathbf{b}(t),\, \mathbf{u}(t) \,\Big)
\end{aligned}$$

En tenant compte des dépendances conditionnelles, nous avons donc :

$$\begin{aligned}
\mathrm{T}\Big(s(t{+}1), a(t), s(t)\Big) = \;& P\Big(\, p_r(t{+}1) \,\Big|\, a(t) \,\Big) && \text{Pose du robot} \\
& \cdot P\Big(\, \mathbf{u}(t{+}1) \,\Big|\, p_r(t),\, \mathbf{u}(t) \,\Big) && \text{Utilité des observations} \\
& \cdot P\Big(\, \mathbf{b}(t{+}1) \,\Big|\, \mathbf{b}(t),\, p_r(t{+}1),\, \mathbf{u}(t{+}1) \,\Big) && \text{Distribution sur les objets}
\end{aligned} \tag{2.24}$$

La pose du robot ainsi que l'utilité des observations sont déterministes. Ainsi, étant donné une action $a(t)$ non terminale, la pose du robot $p_r(t{+}1)$ à l'instant suivant y est égale :

$$P\Big(\, P_r(t{+}1){=}a_t \,\Big|\, a_t \,\Big) = 1 \tag{2.25}$$

De même, l'utilité est mise à jour d'après l'équation 2.19 (page 25). Par contre, la distribution de probabilité jointe $\mathbf{b}(t)$ sur les objets n'est pas déterministe et varie en fonction du résultat d'analyse $\mathbf{z}(t{+}1)$ de la photo prise depuis la pose $p_r(t{+}1)$:

$$\begin{aligned}
\mathbf{b}(t{+}1) &\triangleq P\Big(\, \mathbf{o} \,\Big|\, \mathbf{z}(1{:}t{+}1),\, p_r(1{:}t{+}1) \,\Big) \\
&\propto P\Big(\, \mathbf{z}(t{+}1) \,\Big|\, \mathbf{o},\, \mathbf{z}(1{:}t),\, p_r(1{:}t{+}1) \,\Big) \cdot P\Big(\, \mathbf{o} \,\Big|\, \mathbf{z}(1{:}t),\, p_r(1{:}t) \,\Big) \\
&\propto P\Big(\, \mathbf{z}(t{+}1) \,\Big|\, \mathbf{o},\, \mathbf{u}(t{+}1),\, p_r(t{+}1) \,\Big) \cdot \mathbf{b}(t)
\end{aligned} \tag{2.26}$$

où nous retrouvons le modèle de résultat d'analyse d'une photo (équation 2.20, page 26). A chaque résultat d'analyse $\mathbf{z}(t{+}1)$ correspond une seule distribution de probabilité $\mathbf{b}(t{+}1)$. La probabilité d'obtenir une distribution $\mathbf{b}(t{+}1)$ donnée vaut donc la probabilité d'obtenir le

résultat d'analyse $\mathbf{z}(t{+}1)$ correspondant :

$$\begin{aligned} P\Big(\, \mathbf{b}(t{+}1) \,\Big|\, \mathbf{b}(t),\, p_r(t{+}1),\, \mathbf{u}(t{+}1) \,\Big) \\ &= P\Big(\, \mathbf{z}(t{+}1) \,\Big|\, \mathbf{b}(t),\, p_r(t{+}1),\, \mathbf{u}(t{+}1) \,\Big) \\ &= \sum_{\mathbf{o}} \left[P\Big(\, \mathbf{o},\, \mathbf{z}(t{+}1) \,\Big|\, \mathbf{b}(t),\, p_r(t{+}1),\, \mathbf{u}(t{+}1) \,\Big) \right] \\ &= \sum_{\mathbf{o}} \left[\mathbf{b}(t) \cdot P\Big(\, \mathbf{z}(t{+}1) \,\Big|\, \mathbf{o},\, \mathbf{u}(t{+}1),\, p_r(t{+}1) \,\Big) \right] \end{aligned} \tag{2.27}$$

avec la probabilité jointe sur les cellules $\mathbf{o}$ des objets étant donné la distribution correspondante $\mathbf{b}(t)$ étant évidemment cette même distribution :

$$P\Big(\, \mathbf{o} \,\Big|\, \mathbf{b}(t) \,\Big) = \mathbf{b}(t)$$

Par substitution des équations 2.25, 2.19 et 2.27 dans l'équation 2.24, la fonction de transition T est totalement définie.

CHAPITRE 3 DÉTAILS DE LA SOLUTION

Dans ce chapitre, nous présentons les détails de la résolution du MDP défini au chapitre précédent. Nous commençons par présenter un filtre à particules permettant une estimation rapide de l'état du système à chaque analyse d'une photo. Nous proposons ensuite une politique heuristique du MDP. Nous présentons finalement la méthode MCTS, d'abord de manière générale, puis appliquée au problème étudié.

3.1 Estimation d'état

Dans cette section, nous présentons l'utilisation d'un filtre à particules pour approximer la distribution de probabilité jointe $\mathbf{b}$ sur les cellules des objets. Notre approche est basée sur la méthode de *Sequential Importance Sampling* (SIS) présentée en détails dans le tutoriel de Doucet and Johansen (2009).

3.1.1 Filtre à particules pondérées

Nous approximons la distribution de probabilité jointe $\mathbf{b}$ sur les cellules $\mathbf{o}$ des objets par un ensemble de n_p particules pondérées $\mathbf{o}^i$ telles que :

$$P\Big(\,\mathbf{O}{=}\mathbf{o}\,\Big|\,\mathbf{z}(1{:}t),\,p_r(1{:}t)\,\Big) \approx \sum_{i=1}^{n_p}\Big[\,w_i(t)\cdot\delta\Big(\mathbf{o}^i-\mathbf{o}\Big)\,\Big] \quad \text{avec} \quad \sum_{n=1}^{n_p} w_i(t)=1 \tag{3.1}$$

avec $w_i(t)$ le poids de la particule $\mathbf{o}^i$ à l'instant t. À chaque pas de temps, avec le résultat d'analyse $\mathbf{z}(t{+}1)$ de la dernière photo, ce poids $w_i(t)$ est mis à jour d'après le modèle de résultat d'analyse d'une photo (équation 2.20, page 26) :

$$w_i(t{+}1) \propto P\Big(\,\mathbf{z}(t{+}1)\,\Big|\,\mathbf{O}{=}\mathbf{o}^i,\,\mathbf{u}(t{+}1),\,p_r(t{+}1)\,\Big)\cdot w_i(t) \tag{3.2}$$

On peut montrer que les particules pondérées par les poids $w_i(t{+}1)$ mis à jour approximent effectivement la densité de probabilité jointe $\mathbf{b}(t{+}1)$ sur les cellules $\mathbf{o}$ des objets étant donné

le résultat d'analyse supplémentaire :

$$
\begin{aligned}
&P\Big(\,\mathbf{O}{=}\mathbf{o}\,\Big|\,\mathbf{z}(1{:}t{+}1),\,p_r(1{:}t{+}1)\,\Big) \\
&\qquad\propto P\Big(\,\mathbf{z}(t{+}1)\,\Big|\,\mathbf{O}{=}\mathbf{o},\,\mathbf{z}(1{:}t),\,p_r(1{:}t{+}1)\,\Big)\cdot P\Big(\,\mathbf{O}{=}\mathbf{o}\,\Big|\,\mathbf{z}(1{:}t),\,p_r(1{:}t)\,\Big) \\
&\qquad\propto\!\!\propto P\Big(\,\mathbf{z}(t{+}1)\,\Big|\,\mathbf{O}{=}\mathbf{o},\,\mathbf{z}(1{:}t),\,p_r(1{:}t{+}1)\,\Big)\cdot\sum_{i=1}^{n_p}\Big[\,w_i(t)\cdot\delta\Big(\mathbf{o}^i-\mathbf{o}\Big)\,\Big] \\
&\qquad\propto\!\!\propto \sum_{i=1}^{n_p}\Big[\,\underbrace{P\Big(\,\mathbf{z}(t{+}1)\,\Big|\,\mathbf{O}{=}\mathbf{o}^i,\,\mathbf{u}(t{+}1),\,p_r(t{+}1)\,\Big)\cdot w_i(t)}_{w_i(t+1)}\cdot\delta\Big(\mathbf{o}^i-\mathbf{o}\Big)\,\Big]
\end{aligned}
$$

3.1.2 Dégénérescence et nombre effectif de particules

Après plusieurs prises de mesures, on peut rapidement se trouver dans une situation où seulement un nombre restreint de particules ont un poids significatif. C'est le problème de la **dégénérescence**. Notamment, si le n-ième objet est localisé avec certitude dans une cellule c, seules les quelques particules dont la n-ième composante $\mathbf{o}_n^i$ est c auront un poids non-nul.

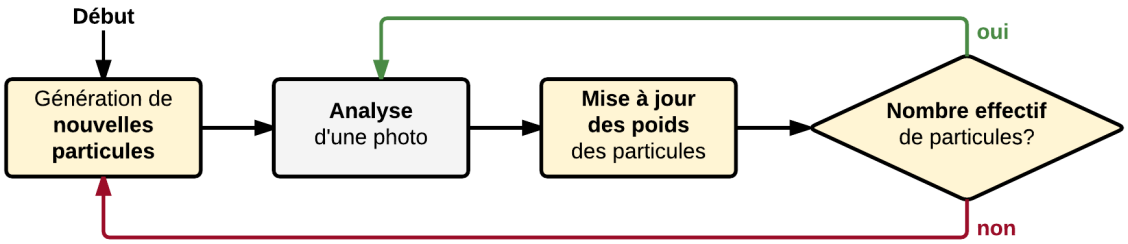


Figure 3.1 **Filtre à particules**. À chaque prise de mesure, le poids des particules est mis à jour. Si le nombre effectif de particules est insuffisant, de nouvelles particules doivent être générées.

Comme illustré à la figure 3.1, pour pallier ce problème nous générons régulièrement de nouvelles particules. Pour décider à quel moment générer de nouvelles particules, nous nous basons sur le **nombre effectif de particules** $n_e(t) \in \mathbb{R}$ à l'instant t défini comme :

$$
n_e(t) = \left(\sum_{i=1}^{n_p}\Big(w_i(t)\Big)^2\right)^{-1} \tag{3.3}
$$

Lorsque ce nombre effectif de particules descend en-dessous d'un seuil donné, nous générons alors un nouvel ensemble de n_p particules équiprobables :

$$n_e(t) < \alpha \cdot n_p \tag{3.4}$$

Ce paramètre $\alpha \in [0, 1]$ permet d'ajuster la fréquence de génération de nouvelles particules. À l'extrême, pour $\alpha = 1$, de nouvelles particules sont générées dès que les particules existantes ne sont plus équiprobables. Pour $\alpha = 0$, de nouvelles particules ne sont jamais générées.

3.1.3 Génération de nouvelles particules

À un moment t donné, on cherche à générer un ensemble de n_p particules $\mathbf{o}^i$ équiprobables approximant la distribution de probabilité jointe sur les cellules $\mathbf{o}_{1:n_i}$ des objets étant donné les instructions $\mathbf{i}_{1:n_i}$ et les résultats d'analyse de photos $\mathbf{z}_{1:n_i}(1{:}t)$ obtenus préalablement :

$$P\Big(\mathbf{o}_{1:n_i} \,\Big|\, \mathbf{o}_0,\, \mathbf{i}_{1:n_i},\, \mathbf{z}_{1:n_i}\Big) \approx \frac{1}{n_p} \cdot \sum_{i=1}^{n_p} \delta\Big(\mathbf{o}^i - \mathbf{o}\Big)$$

où, pour simplifier les notations, nous ne notons pas les indices temporels. Ce problème peut être représenté sous forme d'un réseau de Bayes (figure 3.2) dans lequel les dépendances conditionnelles entre variables connues (cellule initiale $\mathbf{o}_0$, instructions $\mathbf{i}_{1:n_i}$, résultats d'analyses $\mathbf{z}_{1:n_i}$) et inconnues (cellules $\mathbf{o}_{1:n_i}$ des objets) sont données par les modèles stochastiques définis précédemment :

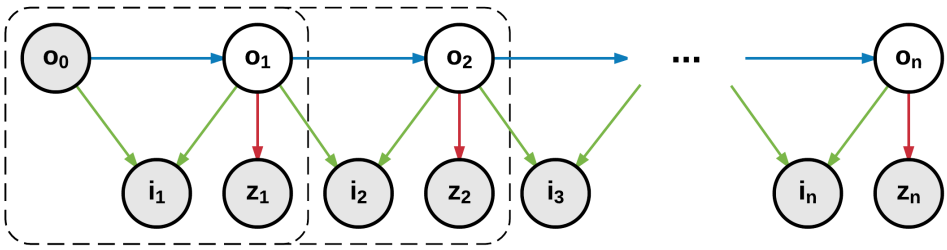


Figure 3.2 **Réseau de Bayes**. Les cellules grisées représentent des variables connues. Les probabilités conditionnelles sont données par le *modèle du détecteur d'objet* (rouge), le *modèle d'instruction* (vert) et le *modèle de contournement* (bleu). Les pointillés représentent les itérations successives de génération.

1. Le **modèle de contournement** (équation 2.7, page 16), donne la distribution de probabilité sur la cellule $\mathbf{o}_n$ d'un objet étant donné la cellule $\mathbf{o}_{n-1}$ de l'objet précédent :

$$P\Big(\, \mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1} \,\Big) \tag{3.5}$$

2. Le **modèle d'instruction** (équation 2.6, page 15) donne la probabilité des indications $\mathbf{i}_n$ du guide étant donné les cellules $\mathbf{o}_n$ et $\mathbf{o}_{n-1}$ de deux objets successifs :

$$P\Big(\, \mathbf{i}_n \,\Big|\, \mathbf{o}_{n-1},\, \mathbf{o}_n \,\Big) \tag{3.6}$$

3. Le **modèle de détecteur d'objet** (équation 2.20, page 26) donne la probabilité d'un résultat d'analyse $\mathbf{z}_n(t)$ pour l'objet n étant donné sa cellule $\mathbf{o}_n$. Avec l'hypothèse d'indépendance des résultats d'analyse, nous pouvons disposer de la probabilité conditionnelle de tous les résultats d'analyse $\mathbf{z}_n(1{:}t)$ jusqu'à l'instant t :

$$P\Big(\, \mathbf{z}_n \,\Big|\, \mathbf{o}_n \,\Big) = P\Big(\, \mathbf{z}_n(1{:}t) \,\Big|\, \mathbf{o}_n \,\Big) = \prod_{l=0}^{t} P\Big(\, \mathbf{z}_n(l) \,\Big|\, \mathbf{o}_n \,\Big) \tag{3.7}$$

Expression récursive de la distribution jointe

Commençons pas observer que la distribution de probabilité jointe sur les n premiers objets (n_i pour tous les objets) se décompose de la manière suivante :

$$\underbrace{P\Big(\, \mathbf{o}_{1:n} \,\Big|\, \mathbf{o}_0,\, \mathbf{i}_{1:n},\, \mathbf{z}_{1:n} \,\Big)}_{\text{Distribution sur } n \text{ objets}} = P\Big(\, \mathbf{o}_{1:n-1} \,\Big|\, \mathbf{o}_0,\, \mathbf{i}_{1:n},\, \mathbf{z}_{1:n} \,\Big) \cdot P\Big(\, \mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1},\, \mathbf{i}_n,\, \mathbf{z}_n \,\Big) \tag{3.8}$$

où la distribution de probabilité sur les $n-1$ premiers objets étant donné les instructions et résultats d'analyse associés aux n premiers objets se décompose encore sous la forme :

$$P\Big(\, \mathbf{o}_{1:n-1} \,\Big|\, \mathbf{o}_0,\, \mathbf{i}_{1:n},\, \mathbf{z}_{1:n} \,\Big) \propto P\Big(\, \mathbf{i}_n,\, \mathbf{z}_n \,\Big|\, \mathbf{o}_{n-1} \,\Big) \cdot \underbrace{P\Big(\, \mathbf{o}_{1:n-1} \,\Big|\, \mathbf{o}_0,\, \mathbf{i}_{1:n-1},\, \mathbf{z}_{1:n-1} \,\Big)}_{\text{Distribution sur } n-1 \text{ objets}} \tag{3.9}$$

Par substitution 3.9 dans 3.8, nous constatons qu'il est possible d'obtenir de manière récursive la distribution de probabilité jointe sur les cellules des n premiers objets à partir de la distribution sur les cellules des $n-1$ premiers objets. Dans l'équation 3.8, il nous faut alors la distribution de probabilité marginale sur la cellule $\mathbf{o}_n$ d'un objet étant donné la cellule $\mathbf{o}_{n-1}$

de l'objet précédent et l'instruction $\mathbf{i}_n$ et les résultats d'analyse $\mathbf{z}_n$ associés :

$$\begin{aligned} P\Big(\mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1},\, \mathbf{i}_n,\, \mathbf{z}_n\Big) &\propto P\Big(\mathbf{i}_n,\, \mathbf{z}_n \,\Big|\, \mathbf{o}_n,\, \mathbf{o}_{n-1}\Big) \cdot P\Big(\mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1}\Big) \\ &\propto \underbrace{P\Big(\mathbf{i}_n \,\Big|\, \mathbf{o}_{n-1},\, \mathbf{o}_n\Big)}_{\text{Instruction}} \cdot \underbrace{P\Big(\mathbf{z}_n \,\Big|\, \mathbf{o}_n\Big)}_{\text{Détecteur}} \cdot \underbrace{P\Big(\mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1}\Big)}_{\text{Contournement}} \end{aligned} \tag{3.10}$$

Dans l'équation 3.9, il nous faut également la probabilité d'une instruction $\mathbf{i}_n$ et des résultats d'analyse $\mathbf{z}_n$ associés à un objet étant donné la cellule de l'objet précédent $\mathbf{o}_{n-1}$:

$$\begin{aligned} P\Big(\mathbf{i}_n,\, \mathbf{z}_n \,\Big|\, \mathbf{o}_{n-1}\Big) &= \sum_{\mathbf{o}_n} \Big[P\Big(\mathbf{o}_n,\, \mathbf{i}_n,\, \mathbf{z}_n \,\Big|\, \mathbf{o}_{n-1}\Big) \Big] \\ &= \sum_{\mathbf{o}_n} \Big[P\Big(\mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1}\Big) \cdot P\Big(\mathbf{i}_n,\, \mathbf{z}_n \,\Big|\, \mathbf{o}_n,\, \mathbf{o}_{n-1}\Big) \Big] \\ &= \sum_{\mathbf{o}_n} \Big[\underbrace{P\Big(\mathbf{i}_n \,\Big|\, \mathbf{o}_{n-1},\, \mathbf{o}_n\Big)}_{\text{Instruction}} \cdot \underbrace{P\Big(\mathbf{z}_n \,\Big|\, \mathbf{o}_n\Big)}_{\text{Détecteur}} \cdot \underbrace{P\Big(\mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1}\Big)}_{\text{Contournement}} \Big] \end{aligned} \tag{3.11}$$

Nous retrouvons finalement dans les équations 3.10 et 3.11 les expressions 3.5, 3.6, et 3.7 qui sont connues en tout temps avec les modèles stochastiques de contournement, de détecteur d'objet et d'instruction définis précédemment.

Génération itérative des particules

Comme illustré à la figure 3.3, nous appliquons une approche itérative basée sur la méthode de *Sequential Importance Sampling* pour générer et rééchantillonner les particules composante par composante. Nous commençons ainsi par **générer la première composante** $\mathbf{o}_1^i$ de chaque particule $\mathbf{o}^i$ en l'échantillonnant depuis la distribution de probabilité sur la cellule $\mathbf{o}_1$ du premier objet étant donné l'instruction et les résultats d'analyse associés (équation 3.10) :

$$\mathbf{o}_1^i \sim P\Big(\mathbf{o}_1 \,\Big|\, \mathbf{o}_0,\, \mathbf{i}_1,\, \mathbf{z}_1\Big) \propto P\Big(\mathbf{i}_1 \,\Big|\, \mathbf{o}_0,\, \mathbf{o}_1\Big) \cdot P\Big(\mathbf{z}_1 \,\Big|\, \mathbf{o}_1\Big) \cdot P\Big(\mathbf{o}_1 \,\Big|\, \mathbf{o}_0\Big) \tag{3.12}$$

Nous obtenons un ensemble de n_p particules partielles qui n'ont donc qu'une seule composante. Nous allons ensuite **rééchantillonner ces particules partielles** pour prendre en compte l'instruction et les résultats d'analyses relatifs au second objet. Plus précisément, sur la base de l'équation 3.10, nous associons un poids w_i à chaque particule partielle $\mathbf{o}_1^i$:

$$w_i \propto P\Big(\mathbf{i}_2,\, \mathbf{z}_2 \,\Big|\, \mathbf{o}_1^i\Big) = \sum_{\mathbf{o}_2} \Big[P\Big(\mathbf{i}_2 \,\Big|\, \mathbf{o}_1^i,\, \mathbf{o}_2\Big) \cdot P\Big(\mathbf{z}_2 \,\Big|\, \mathbf{o}_2\Big) \cdot P\Big(\mathbf{o}_2 \,\Big|\, \mathbf{o}_1^i\Big) \Big] \tag{3.13}$$

Figure 3.3 **Génération itérative de particules** par alternance des deux étapes de génération de composantes et d'échantillonnage des particules partielles jusqu'à obtention de particules complètes (équation 3.1). Observons que l'étendue de la distribution marginale sur chaque objet diminue après chaque rééchantillonnage, contrainte par la prise en compte de l'instruction et des résultats d'analyse relatifs à un objet supplémentaire.

D'après l'équation 3.9, en rééchantillonnant les particules partielles d'après cette distribution des poids (*importance resampling*), nous obtenons un nouvel ensemble de particules approximant la distribution de probabilité sur la cellule $\mathbf{o}_1$ du premier objet étant donné les instructions et résultats d'analyses relatifs aux premier *et* second objets :

$$P\Big(\,\mathbf{o}_1\,\Big|\,\mathbf{o}_0,\,\mathbf{i}_1,\,\mathbf{z}_1\,\Big)\quad\xrightarrow[\textit{rééchantillonnage}]{}\quad P\Big(\,\mathbf{o}_1\,\Big|\,\mathbf{o}_0,\,\mathbf{i}_1,\,\mathbf{z}_1,\,\mathbf{i}_2,\,\mathbf{z}_2\,\Big)$$

Comme nous l'avons fait pour la première composante (équation 3.12), nous pouvons alors générer la seconde composante $\mathbf{o}_2^i$ de chaque particule $\mathbf{o}^i$ en l'échantillonnant depuis la distribution de probabilité sur la cellule $\mathbf{o}_2$ du second objet étant donné la première composante $\mathbf{o}_1^i$ et l'instruction et les résultats d'analyse associés (équation 3.10) :

$$\mathbf{o}_2^i \sim P\Big(\,\mathbf{o}_2\,\Big|\,\mathbf{o}_1^i,\,\mathbf{i}_2,\,\mathbf{z}_2\,\Big) \propto P\Big(\,\mathbf{i}_2\,\Big|\,\mathbf{o}_1^i,\,\mathbf{o}_2\,\Big)\cdot P\Big(\,\mathbf{z}_2\,\Big|\,\mathbf{o}_2\,\Big)\cdot P\Big(\,\mathbf{o}_2\,\Big|\,\mathbf{o}_1^i\,\Big)$$

Nous obtenons alors un ensemble de n_p particules partielles de deux composantes qui approximent, conformément à l'équation 3.8, la distribution de probabilité jointe sur les cellules des deux premiers objets étant donné les instructions et résultats d'analyse associés. De nouveau, nous pouvons associer un poids w_i à chacune de ces particules partielles (équation 3.10) :

$$w_i \propto P\Big(\,\mathbf{i}_3,\,\mathbf{z}_3\,\Big|\,\mathbf{o}_2^i\,\Big) = \sum_{\mathbf{o}_3}\Bigg[\,P\Big(\,\mathbf{i}_3\,\Big|\,\mathbf{o}_2^i,\,\mathbf{o}_3\,\Big)\cdot P\Big(\,\mathbf{z}_3\,\Big|\,\mathbf{o}_3\,\Big)\cdot P\Big(\,\mathbf{o}_3\,\Big|\,\mathbf{o}_2^i\,\Big)\,\Bigg]$$

et les rééchantillonner d'après la distribution de ces poids pour prendre en compte l'instruction et les résultats d'analyses relatifs au troisième objet :

$$P\Big(\,\mathbf{o}_{1:2}\,\Big|\,\mathbf{o}_0,\,\mathbf{i}_{1:2},\,\mathbf{z}_{1:2}\,\Big)\quad\xrightarrow[\textit{rééchantillonnage}]{}\quad P\Big(\,\mathbf{o}_{1:2}\,\Big|\,\mathbf{o}_0,\,\mathbf{i}_{1:3},\,\mathbf{z}_{1:3}\,\Big)$$

Il est alors possible de générer la troisième composante de chaque particule, de les rééchantillonner pour prendre en compte l'instruction et les résultats d'analyse relatifs au quatrième objet, puis de générer la quatrième composante, et ainsi de suite jusqu'à l'obtention de particules complètes avec la génération des dernières composantes. En résumé, cette méthode de génération de particules consiste donc en l'application itérative de deux étapes :

1. **Générer une composante**. Nous générons la n-ième composante $\mathbf{o}_n^i$ de chaque particule $\mathbf{o}^i$ en l'échantillonnant depuis la distribution de probabilité sur la cellule $\mathbf{o}_n$ du n-ième objet étant donné la composante précédente $\mathbf{o}_{n-1}^i$ et l'instruction $\mathbf{i}_n$ et les résultats d'analyse $\mathbf{z}_n$ associés :

$$\mathbf{o}_n^i \sim P\Big(\, \mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1}^i,\, \mathbf{i}_n,\, \mathbf{z}_n \,\Big) \propto \underbrace{P\Big(\, \mathbf{i}_n \,\Big|\, \mathbf{o}_{n-1}^i,\, \mathbf{o}_n \,\Big)}_{\text{Instruction}} \cdot \underbrace{P\Big(\, \mathbf{z}_n \,\Big|\, \mathbf{o}_n \,\Big)}_{\text{Détecteur}} \cdot \underbrace{P\Big(\, \mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1}^i \,\Big)}_{\text{Contournement}} \quad (3.14)$$

2. **Rééchantillonnage des particules partielles**. Nous associons à chaque particule partielle $\mathbf{o}_{1:n-1}^i$ un poids w_i et les rééchantillonnons d'après la distribution de ces poids pour prendre en compte l'instruction et les résultats d'analyses associés à un objet supplémentaire.

$$w_i \propto P\Big(\, \mathbf{i}_n,\, \mathbf{z}_n \,\Big|\, \mathbf{o}_{n-1}^i \,\Big) = \sum_{\mathbf{o}_n} \Bigg[\underbrace{P\Big(\, \mathbf{i}_n \,\Big|\, \mathbf{o}_{n-1}^i,\, \mathbf{o}_n \,\Big)}_{\text{Instruction}} \cdot \underbrace{P\Big(\, \mathbf{z}_n \,\Big|\, \mathbf{o}_n \,\Big)}_{\text{Détecteur}} \cdot \underbrace{P\Big(\, \mathbf{o}_n \,\Big|\, \mathbf{o}_{n-1}^i \,\Big)}_{\text{Contournement}} \Bigg] \quad (3.15)$$

Probabilités des résultats d'analyse et état du système

Observons que, pour cette méthode de génération de nouvelles particules, la probabilité des résultats d'analyse $\mathbf{z}(1{:}t)$ est requise. Nous notons ainsi $\mathbf{m}(t)$ ces probabilités sous forme matricielle, avec la composante $\mathbf{m}_{n,c}(t)$ la probabilité des résultats d'analyse pour l'objet n étant donné que l'objet se trouve dans la cellule c :

$$\mathbf{m}_{n,c}(t) = P\Big(\, \mathbf{z}_n(1{:}t) \,\Big|\, \mathbf{O}_n{=}c,\, p_r(1{:}t) \,\Big) = \prod_{l=0}^{t} P\Big(\, \mathbf{z}_n(l) \,\Big|\, \mathbf{O}_n{=}c,\, p_r(l) \,\Big)$$

Nous ajoutons ces probabilités à l'état du système (équation 2.21, page 28) :

$$s(t) \triangleq \Big\{\, p_r(t),\, \mathbf{b}(t),\, \mathbf{u}(t),\, \mathbf{m}(t) \,\Big\}$$

Elles sont aisément mises à jour après chaque photo à l'aide du modèle de résultat d'analyse :

$$\mathbf{m}_{n,c}(t{+}1) = \mathbf{m}_{n,c}(t) \cdot P\Big(\, \mathbf{z}_n(t{+}1) \,\Big|\, \mathbf{O}_n{=}c,\, p_r(t{+}1),\, \mathbf{u}(t{+}1) \,\Big)$$

Et il est ainsi possible de générer de nouvelles particules à tout instant t. Observons que pour la génération initiale de particules, avant la première analyse de photo, cette probabilité $\mathbf{m}_{n,c}(t)$ est unitaire pour toutes les cellules et tous les objets.

3.2 Contrôle de haut niveau

Considérons l'étudiant perdu de l'introduction : pour décider du chemin à prendre, il ne va certainement pas évaluer à chaque pas l'*intérêt de poser son pied un peu plus à gauche ou un peu plus à droite*. Par contre, il est plus naturel pour lui de se fixer des objectifs en décidant, par exemple, *d'aller voir au bout du couloir s'il y voit les machines distributrices*. Pendant qu'il se déplace, il remet de temps en temps son objectif en question d'après ce qu'il observe de l'environnement ; par exemple s'il trouve les machines distributrices avant d'atteindre son objectif au bout du couloir. Ce comportement est illustré à la figure 3.4

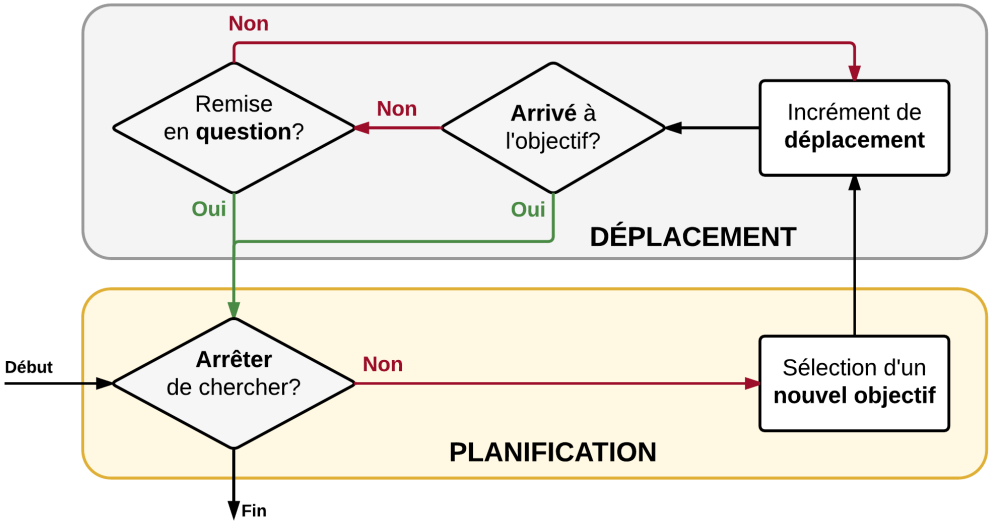


Figure 3.4 **Planification de haut niveau**. Le robot se déplace vers un objectif qu'il peut remettre en question en fonction de ce qu'il observe en chemin de l'environnement.

Dans cette optique, plutôt que de décider à chaque instant du prochain déplacement élémentaire du robot, nous proposons un contrôle à un niveau d'abstraction supérieur. Comme illustré à la figure 3.4, la **planification à haut niveau** consiste à décider, tant que la recherche continue, d'une pose navigable de l'environnement comme objectif du robot. Celui-ci décide ensuite d'un chemin jusqu'à cet objectif en utilisant une méthode classique de planification de trajectoire comme, par exemple, Dijkstra (1959). Après chaque déplacement élémentaire, il décide alors s'il est nécessaire de remettre son objectif en question en fonction du résultat d'analyse de la dernière photo.

Remise en question d'un objectif

Typiquement, on imagine que l'étudiant perdu remet son objectif en question si, par exemple, il repère l'objet dont il est à la recherche avant d'atteindre son objectif, ou encore si, d'après ce qu'il voit en chemin, l'objet ne semble pas se trouver là où il comptait regarder. Plus formellement, dès que ce qu'il connaît de l'environnement change significativement par rapport au moment où il s'était fixé un objectif, il remet celui-ci en question.

Dans notre problème, ce que le robot *connaît de l'environnement* est la distribution de probabilité jointe $\mathbf{b}(t)$ sur les objets recherchés. Pour évaluer quand cette distribution change de manière significative, nous utilisons de nouveau le nombre effectif de particules $n_e(t)$, introduit plus haut dans le cadre de la génération de nouvelles particules (équation 3.3). Ainsi, si n_o est le nombre effectif de particules lors de la sélection du dernier objectif, on remet celui-ci en question dès que le nombre effectif $n_e(t)$ de particules passe en dessous d'un certain seuil :

$$n_e(t) \leq \beta \cdot n_o \tag{3.16}$$

avec $\beta \in [0, 1]$ un paramètre qui permet de régler une certaine *sensibilité au changement* : pour $\beta = 1$, le moindre changement de la distribution entraîne une remise en question ; pour $\beta = 0$, le robot avance toujours jusqu'à l'objectif.

Remise en question lors de la génération Puisque le nombre effectif de particules y est réinitialisé, nous remettons également l'objectif en question lors de chaque génération de nouvelles particules d'après le critère 3.4. Observons que les critères de remise en question 3.16 et de génération de particules 3.4 sont remplis en même temps pour un nombre effectif de particules initial n_o suffisant :

$$n_o > \frac{\alpha}{\beta} \cdot n_p$$

Dès lors, tant que la paramètre α est significativement plus petit que le paramètre β, le nombre de remises en questions excédentaires est faible et les générations de nouvelles particules sont significativement moins fréquentes que les remises en question de l'objectif. Une définition plus complexe du critère 3.16 est envisageable pour prendre en compte la génération et éliminer ces remises en questions excédentaires, mais cette solution simple suffit amplement dans notre cas.

3.3 Politique heuristique

Dans cette section, nous proposons une politique heuristique pour la résolution du problème. Comme nous le verrons dans la section suivante, elle nous servira comme *politique par défaut* dans l'application de la méthode MCTS. En partant du contrôle de haut niveau introduit à la section précédente, il reste à maintenant à déterminer, d'une part, comment est-ce qu'on sélectionne un objectif et, d'autre part, quand est-ce qu'on décide d'arrêter de chercher.

3.3.1 Sélection d'un objectif

Observons que le problème de la sélection d'un objectif présente deux facettes :

1. D'une part, il s'agit d'abord de décider **quel objet chercher**. On pourrait simplement se contenter de localiser les objets successivement : on passe à la recherche de l'objet suivant que lorsqu'on est sûr de la position de l'objet courant. Mais on pourrait également envisager de passer l'un ou l'autre objet, si par exemple, il est dur à détecter ou encore si sa découverte ne contribuerait que peu à l'objectif global de localiser le dernier objet.

2. D'autre part, il faut ensuite décider **où se rendre** pour essayer de localiser l'objet recherché : à partir de quelle pose devrait-on prendre une photo en vue de localiser rapidement et précisément l'objet recherché ?

Nous proposons ainsi une politique basée sur la maximisation d'une combinaison de deux fonctions heuristiques : l'utilité des objets et l'utilité des poses.

Erreur attendue et utilité d'un objet

Étant donné la distribution de probabilité marginale $\mathbf{b}_n$ sur la cellule du n-ième objet, nous commençons par définir l'**erreur attendue** A, une mesure de la dispersion de cette distribution :

$$\mathrm{A}\Big(\mathbf{b}_n\Big) \triangleq \sum_{o \in \mathcal{O}^C} \sum_{c \in \mathcal{O}^C} \left[\mathbf{b}_{n,o} \cdot \mathbf{b}_{n,c} \cdot \mathrm{D}_c\Big(o,\, c\Big) \right] \tag{3.17}$$

avec D_c la distance de contournement entre deux cellules. Il s'agit d'une version adaptée de la moyenne des différences absolues (*Mean Absolute Difference*, Yitzhaki, 2003). L'erreur attendue $\mathrm{A}(\mathbf{b}_n)$ donne ainsi la distance de contournement espérée entre la cellule contenant

réellement le n-ième objet et une autre cellule quelconque échantillonnée depuis la distribution $\mathbf{b}_n$. Observons que l'erreur attendue sera d'autant plus grande que la distribution est segmentée par des cellules occupées de l'environnement. Par exemple, si la distribution est partagée entre deux couloirs, les grandes distances de contournement pour se rendre des cellules d'un couloir aux cellules de l'autre auront un impact significatif sur l'erreur attendue.

Intuitivement, dans notre objectif global de déterminer la cellule $\mathbf{o}_{n_i}$ de l'objet final, localiser un objet intermédiaire n'est réellement utile que si cela permet de réduire l'erreur attendue pour l'objet final. Par exemple, il est inutile d'essayer de localiser un objet intermédiaire au milieu d'un couloir si le robot sait déjà que *l'objet final se trouve de toute façon au bout* de ce couloir. Au contraire, s'il y a deux couloirs parallèles entre lesquels le robot hésite, localiser l'objet intermédiaire permet de faire plus rapidement un choix entre ceux-ci.

En ce sens, nous définissons l'**utilité de l'objet** n comme la diminution espérée de l'erreur attendue pour l'objet final n_i dans la situation hypothétique où l'on déterminerait avec certitude la cellule $\mathbf{o}_n$ contenant cet objet intermédiaire :

$$\mathrm{U}_o\big(n, \mathbf{b}\big) \triangleq \sum_{o \in \mathcal{O}^C} \left[\mathbf{b}_{n,o} \cdot \Big(\mathrm{A}\big(\mathbf{b}_{n_i}\big) - \mathrm{A}\big(\mathbf{b}_{n_i} \,|\, \mathbf{O}_n{=}o \big) \Big) \right] \tag{3.18}$$

où nous notons $(\mathbf{b}_{n_i} \,|\, \mathbf{O}_n{=}o)$ la distribution de probabilité marginale sur le dernier objet n_i étant donné la cellule o de l'objet n :

$$(\mathbf{b}_{n_i} \,|\, \mathbf{O}_n{=}o) \triangleq P\Big(\mathbf{O}_{n_i} \,\Big|\, \mathbf{O}_n{=}o,\, \mathbf{z}(1{:}t),\, p_r(1{:}t) \Big)$$

Notons qu'avec l'approximation de la distribution jointe $\mathbf{b}$ par un ensemble de particules (équations 3.1), cette distribution conditionnelle s'obtient simplement en ne conservant que le sous-ensemble des particules passant par la cellule o considérée :

$$P\Big(\mathbf{o}_{n_i} \,\Big|\, \mathbf{o}_n,\, \mathbf{z}(1{:}t),\, p_r(1{:}t) \Big) \propto\!\!\!\propto \delta\Big(\mathbf{o}_n^i - \mathbf{o}_n\Big) \cdot \sum_{i=1}^{n_p} \Big[w_i(t) \cdot \delta\Big(\mathbf{o}_{n_i}^i - \mathbf{o}_{n_i}\Big) \Big]$$

Probabilité de détection et utilité d'une pose

Intuitivement, une photo prise par le robot est d'autant plus utile que la probabilité d'y détecter l'objet est importante. Selon que l'objet soit détecté ou non, la photo permet de

confirmer ou d'infirmer sa présence dans le champ de vision. Autrement dit, prendre une photo d'un endroit où on pense que l'objet ne se trouve pas n'a aucun intérêt.

La probabilité de détecter l'objet n depuis une pose p_r est donnée par la somme sur les cellules utiles $\mathcal{U}$ du champ de vision $\mathcal{V}_p$ (équation 2.20, page 26) de la distribution de probabilité sur la cellule o_n contenant l'objet, pondérée par la probabilité de détection (équation 2.10, 20) de l'objet dans chacune de ces cellules :

$$\sum_{c \in \mathcal{U}} \Big[\mathbf{b}_{n,c} \cdot P\Big(\mathbf{D}_n \,\Big|\, \mathbf{O}_n{=}c,\, p \Big) \Big]$$

avec l'événement $\mathbf{D}_n \triangleq$ *« Détecter correctement l'objet n dans la photo »*. Ainsi, de manière similaire à ce qui a été proposé par Shubina and Tsotsos (2010), nous définissons l'**utilité d'une pose** p pour la recherche de l'objet n comme la probabilité d'y détecter cet objet sur le temps de parcours pour se rendre jusqu'à cette pose :

$$\mathrm{U}_p\Big(p, p_r, \mathbf{b}_n\Big) \triangleq \frac{1}{\mathrm{T}_p(p_r, p)} \cdot \sum_{c \in \mathcal{U}} \Big[\mathbf{b}_{n,c} \cdot P\Big(\mathbf{D}_n \,\Big|\, \mathbf{O}_n{=}c,\, p_r \Big) \Big] \tag{3.19}$$

avec p_r la pose actuelle du robot et $\mathrm{T}_p(p_r, p)$ le temps de parcours de la pose p_r à la pose p (section 2.1). Considérer le temps de parcours au dénominateur favorise les poses plus proches du robot. Intuitivement, on peut comprendre cette approche comme cherchant à maximiser, à court terme, la probabilité de détecter l'objet par *unité de temps passé à le chercher*.

Sélection d'une pose comme objectif

Finalement, nous définissons l'**utilité globale d'une pose** p comme la somme de l'utilité de cette pose à la recherche de chaque objet n (équation 3.19) pondérée par les utilités respectives de ces objets (équation 3.18) :

$$\mathrm{U}\Big(p, p_r, \mathbf{b}\Big) \triangleq \sum_{n=1}^{n_i} \Big(\mathrm{U}_p\Big(p, p_r, \mathbf{b}_n\Big) \cdot \mathrm{U}_o\Big(n, \mathbf{b}\Big) \Big) \tag{3.20}$$

Notons que, à moins que deux objets soient particulièrement proches, l'utilité d'une pose p donnée n'est généralement significative que pour un seul objet (et donc un seul terme de

cette somme). La sélection d'une nouvelle pose p comme objectif se fait alors simplement en maximisant cette utilité globale (équation 3.20) :

$$p = \underset{p}{\operatorname{argmax}} \Big(\mathrm{U}\big(p, p_r, \mathbf{b}\big) \Big)$$

3.3.2 Critère de terminaison

Avant de sélectionner un nouvel objectif, le robot doit décider s'il continue la recherche où s'il sélectionne l'action terminale. Un critère de terminaison simple consiste à continuer à chercher jusqu'à ce que la probabilité de localiser correctement le dernier objet soit supérieure à un seuil donné :

$$\max_c \Big(\mathbf{b}_{n_i,c} \Big) > \gamma$$

avec $\gamma \in [0, 1]$ suffisamment grand pour éviter que le robot s'arrête de chercher prématurément, mais pas trop grand au cas où ce dernier objet serait difficile à localiser (typiquement, $\gamma \approx 0.8$). En plus de ce critère, pour assurer un temps d'exécution fini, nous imposons également une limite supérieure de temps à la recherche.

3.4 Monte Carlo Tree Search (MCTS)

Dans cette section, nous présentons la méthode de *Monte Carlo Tree Search* (MCTS) et son application au problème étudié. Nous commençons par présenter l'algorithme de *rollout* pour la résolution d'un processus décisionnel de Markov tel que défini dans l'introduction (section 1.1.3, page 6). Nous présentons ensuite la méthode UCB (*Upper Confidence Bound*), et son application à la résolution de problèmes du type *multi-armed bandit*, qui nous permettra d'introduire par analogie la méthode MCTS. Nous finirons en présentant les détails de l'application de la méthode MCTS à notre problème.

3.4.1 Simulations de *rollout*

Imaginons que nous disposons initialement d'une politique π_0 non-optimale, appelée **politique par défaut** [1]. Avec la fonction de transition T, nous pouvons ainsi **simuler le système**

1. Également appelée politique de base ou politique de *rollout*. Attention à la confusion : selon les auteurs, le terme *politique de rollout* peut faire référence soit à la politique par défaut, soit à la politique obtenue comme résultat final de l'algorithme de *rollout*.

par échantillonnage de chaque état successif, jusqu'à l'application d'une action terminale :

$$s_{t+1} \sim \mathrm{T}\Big(S_{t+1},\, \pi_0(s_t),\, s_t \Big) \tag{3.21}$$

Pour chaque simulation, appelée **simulation de rollout**, nous faisons la somme des récompenses obtenues. En réalisant un nombre suffisant de simulations depuis un état initial s_t, nous obtenons ainsi une estimation $\bar{v}_0(s_t)$ de la valeur de cet état d'après la politique π_0 (équation 1.1, page 7) comme la moyenne (algorithme 1) :

$$\lim_{n\to\infty} \bar{v}_0(s_t) = \lim_{n\to\infty} \left[\frac{1}{n} \cdot \Big(r_1 + r_2 + \cdots + r_n \Big) \right] = v_0(s_t)$$

avec n et r_i le nombre et les résultats de simulations. Par extension, nous définissons la **valeur d'une action** $v_\pi(s_t, a_t)$ d'après une politique π comme la valeur de l'état s_t si on commence par y appliquer l'action a_t avant d'appliquer la politique π aux états ultérieurs :

$$v_\pi(s_t, a_t) = \mathrm{R}(s_t, a_t) + E\Big(v_\pi(s_{t+1}) \,\Big|\, s_t, a_t \Big)$$

Ainsi, la **valeur optimale d'une action** $v^*(s_t, a_t)$ est la plus grande valeur possible de cette action ; autrement dit, la valeur de cette action si la politique optimale π^* (équation 1.2) est appliquée aux états ultérieurs :

$$v^*(s_t, a_t) = \mathrm{R}(s_t, a_t) + E\Big(v^*(s_{t+1}) \,\Big|\, s_t, a_t \Big)$$

L'expression de l'**action optimale** $\pi^*(s_t)$ à appliquer à l'état s_t (équation 1.3) peut alors être reformulée simplement en terme de sélection de l'action dont la valeur est maximale :

$$\pi^*(s_t) = \underset{a_t \in \mathcal{A}_{s_t}}{\operatorname{argmax}} \Big(v^*(s_t, a_t) \Big) \tag{3.22}$$

De nouveau, en commençant un nombre suffisant de simulations par l'application de l'action a_t à un état initial s_t, nous obtenons une estimation de la valeur $\bar{v}_0(s_t, a_t)$ de l'action a_t à cet état d'après la politique π_0 (algorithme 2).

Algorithme 1 Simulation de rollout

```
1: function ROLLOUT(s)
2:     a ~ DEFAULT-POLICY(s)
3:     if TERMINAL(a) then
4:         return REWARD(s, a)
5:     else
6:         s ~ TRANSITION(s, a)
7:         return REWARD(s, a) + ROLLOUT(s)
8:     end if
9: end function
```

Algorithme 2 Simulation de rollout (avec première action imposée)

```
1: function ROLLOUT-ACTION(s, a)
2:     s ~ TRANSITION(s, a)
3:     return REWARD(s, a) + ROLLOUT(s)
4: end function
```

Notons qu'il est possible de calculer cette estimation de la valeur de manière récursive. En notant $\bar{v}_n$ la valeur estimée après n résultats de simulation, nous avons (*moyenne glissante*) :

$$\begin{aligned}\bar{v}_{n+1} &= \frac{1}{(n+1)} \cdot \Big(r_1 + r_2 + \cdots + r_n + r_{n+1}\Big) \\ &= \frac{1}{(n+1)} \cdot \Big(n \cdot \bar{v}_n + r_{n+1}\Big) \\ &= \bar{v}_n + \frac{1}{(n+1)} \cdot \Big(r_{n+1} - \bar{v}_n\Big) \end{aligned} \tag{3.23}$$

où seule l'estimation de la valeur $\bar{v}_n$ sur la base de n simulations est nécessaire au calcul de l'estimation de la valeur $\bar{v}_{n+1}$ étant donné un résultat de simulation r_{n+1} supplémentaire.

Algorithme de rollout

En remplaçant dans l'équation 3.22 la valeur optimale d'une action par sa valeur approchée $\bar{v}_0(s_t, a_t)$ d'après la politique π_0, nous obtenons directement une **nouvelle politique** π_1 plus proche de la politique optimale π^* (Bertsekas, 2005, p. 338) :

$$\pi_1(s_t) = \underset{a_t \in \mathcal{A}_{s_t}}{\operatorname{argmax}} \Big[\, \bar{v}_0(s_t, a_t) \,\Big] \tag{3.24}$$

Algorithme 3 Algorithme de rollout (non-récursif)

```
1: function ROLLOUT-POLICY(s, n)
2:     A ← ACTIONS(s)
3:     V ← [0 ... 0]
4:     for i = 1 ... n_A do
5:         loop n
6:             V[i] ← V[i] + ROLLOUT-ACTION(s, A[i])
7:         end loop
8:     end for
9:     return A[argmax_i(V[i])]
10: end function
```

Cette opération est la base de l'**algorithme de *rollout*** (Bertsekas, 2005, p. 335) : à un état donné s_t, on sélectionne l'action $\pi_1(s_t)$ à appliquer en estimant par simulation la valeur de chacune des actions applicables d'après une politique par défaut π_0 (algorithme 3). Notons que cet algorithme peut être appliqué de manière récursive dans le but de converger vers la politique optimale :

$$\pi_{k+1}(s_t) = \underset{a_t \in \mathcal{A}_{s_t}}{\operatorname{argmax}} \left[\mathrm{R}\Big(s_t, a_t\Big) + E\Big(\bar{v}_k(s_{t+1}) \,\Big|\, s_t,\, a_t \Big) \right]$$

Politique stochastique

Nous avons jusqu'à maintenant considéré uniquement des politiques déterministes, c'est-à-dire des politiques $\pi : \mathcal{S} \to \mathcal{A}$ qui associent à chaque état du système une action unique à appliquer. Cependant, pour les simulations de *rollout*, nous pourrions également utiliser une politique par défaut stochastique. Ainsi, nous notons $\pi(s_t, a_t)$ la probabilité d'appliquer l'action a_t à l'état s_t d'après la politique stochastique π. Pour simuler le système, comme nous le faisons déjà pour les états (équation 3.21), nous échantillonnons les actions successives a_t depuis la politique stochastique :

$$a_t \sim \pi(s_t, A_t)$$

La politique stochastique la plus simple consiste ainsi à considérer une distribution de probabilité uniforme sur l'ensemble $\mathcal{A}_{s_t}$ des actions applicables à chaque état.

Algorithme 4 Algorithme de rollout (avec politique de sélection)

```
1: function ROLLOUT-POLICY(V, N)
2:     A ← ACTIONS(s)
3:     V ← [0 ... 0]
4:     N ← [0 ... 0]
5:     loop n
6:         i ← SELECTION-POLICY(V, N)
7:         r ← ROLLOUT-ACTION(s, A[i])
8:         N[i] ← N[i] + 1
9:         V[i] ← V[i] + (r − V[i]) /N[i]
10:    end loop
11:    return A[argmax_i(V[i])]
12: end function
```

3.4.2 Politique de sélection des simulations

Étant donné l'état actuel s_t du système, imaginons que nous voulons effectuer un ensemble de simulations de rollout pour estimer les valeurs $\bar{v}(s_t, a_t)$ des différentes actions applicables à partir de cet état. Les ressources disponibles pour ces simulations sont limitées et la question qui se pose alors est de comment répartir le plus efficacement possible ces ressources entres les simulations pour les différentes actions.

Par exemple, imaginons qu'à l'état s_t nous avons le choix entre trois actions a_t^1, a_t^2 et a_t^3 et qu'un quota de maximum 300 simulations nous est imposé. La démarche la plus simple consisterait alors à réaliser 100 simulations pour chaque action et à sélectionner la meilleure action *a posteriori* (équation 3.24). Cette démarche n'est cependant certainement pas optimale. Si, par exemple, la valeur réelle de l'action a_t^1 est largement inférieure aux deux autres, seules quelques simulations devraient suffire à s'en rendre compte à rejeter cette action. Le reste du quota des simulations pourrait ainsi être mieux mis à profit pour distinguer la meilleure action entre a_t^2 et a_t^3.

Nous cherchons donc à définir une **politique de sélection** qui, étant donné les résultats de simulation déjà obtenus, spécifie pour quelle action effectuer la prochaine simulation. L'algorithme 4 illustre l'application d'une politique de sélection à la sélection des simulations dans l'algorithme de *rollout*.

Problème du *multi-armed bandit*

Semblable à ce problème de sélection des simulations, le problème du *multi-armed bandit* est un exemple classique de dilemme entre l'exploration et l'exploitation. On imagine un ensemble de n_b machines à sous (*bandit*). À chaque tour, le joueur tire le levier d'une de ces machines et reçoit une récompense ; on note $i(n)$ l'indice de la machine sélectionnée au tour n et $r(n)$ la récompense correspondante obtenue. Cette récompense suit une distribution de probabilité invariable dans le temps mais distincte pour chaque machine ; on note v_i l'espérance de gain réelle de la machine i, inconnue du joueur. Pendant qu'il joue, le joueur prend note, d'une part, du nombre de fois $n_i(n)$ que chaque machine i a été jouée et, d'autre part, de la moyenne $\bar{v}_i(n)$ des récompenses obtenues sur cette machine jusqu'au n-ième tour compris. Son objectif est de maximiser ses gains à long terme en résolvant le dilemme entre l'exploration et l'exploitation :

1. **Exploitation** : jouer la machine pour laquelle l'espérance de gain $\bar{v}_i(n)$ est la plus élevée d'après les récompenses préalablement obtenues.
2. **Exploration** : essayer les autres machines pour vérifier que leurs espérances de gain v_i sont bien inférieures.

Dans ce contexte, le **regret attendu** $\rho(n)$ après n tours est défini comme la différence entre le gain espéré si le joueur joue toujours la meilleure machine et le gain réel espéré d'après la politique $i(n)$ du joueur :

$$\rho(n) = n \cdot \max_i(v_i) \; - \; E\left(\sum_{t=0}^{n} v_{i(t)}\right)$$

Ce regret attendu est donc représentatif de la perte de gain due au fait que le joueur ne sélectionne pas toujours la meilleur machine. Lai and Robbins (1985) montrent que, quelle que soit la politique du joueur, ce regret attendu ne peut pas croître plus lentement que $\mathcal{O}(\ln(n))$ étant donné certaines conditions sur les distributions des récompenses. La politique $i(n)$ du joueur est alors dite *efficace* si son taux de croissance est égal, à un facteur constant prêt, à ce taux de croissance optimal.

Upper Confidence Bound (UCB)

Auer et al. (2002) proposent une telle politique efficace : UCB. Avec le nombre de fois $n_i(n)$ que chaque machine i a été jouée et la moyenne $\bar{v}_i(n)$ des récompenses obtenues sur cette machine jusqu'au n-ième tour compris, on sélectionne la machine $i(n+1)$ jouée au tour suivant

en maximisant :

$$i(n+1) = \underset{i \in [1, n_a]}{\operatorname{argmax}} \left[\bar{v}_i(n) + \sqrt{\frac{2\ln(n)}{n_i(n)}} \right] \tag{3.25}$$

Intuitivement, on observe que cette méthode permet de faire le compromis entre l'exploitation des machines prometteuses (lorsque le gain moyen $\bar{v}_i(n)$ est élevée) et l'exploration des machines peu jouées (lorsque le nombre $n_i(n)$ est petit). Notons également qu'avec le nombre d'essais $n_i(n)$ au dénominateur, une machine qui n'a jamais été jouée sera toujours sélectionnée en priorité.

L'analogie entre le problème du *multi-armed bandit* et notre problème de sélection des simulations est évidente : tirer le levier de la machine i revient à simuler le système en appliquant l'action a_t^i, le gain r_i correspond au résultat de la simulation et le gain moyen $\bar{v}_i$ à l'estimation $\bar{v}(s_t, a_t^i)$ de la valeur de l'action a_t^i. La méthode UCB peut ainsi être utilisée comme politique de sélection (algorithme 4).

3.4.3 Monte-Carlo Tree Search

Comme illustré à la figure 3.5, nous représentons un MDP sous la forme d'un arbre avec des **nœuds états** et des **nœuds actions** représentant respectivement les états s_t du système et actions a_t applicables. Le nœud racine de l'arbre représente l'état actuel du système. Pour un problème déterministe, cette représentation peut être simplifiée en représentant les états par des nœuds et les actions par une branche entre deux nœuds. Chaque chemin de la racine à une feuille d'un tel arbre représente une séquence différente d'états et d'actions possibles à partir d'un état initial jusqu'à l'application d'une action terminale.

Avec l'algorithme de *rollout*, seule la politique par défaut est utilisée pour évaluer par simulation les valeurs des actions applicables à l'état actuel du système. Avec la méthode de *Monte-Carlo tree search* (Kocsis and Szepesvári, 2006; Browne et al., 2012), ces simulations sont guidées par une politique de sélection dans un arbre, basée sur une estimation de la valeur de chaque nœud rencontré. Cette arbre est construit progressivement, guidé par les résultats des simulations précédentes, et les estimations des valeurs des nœuds deviennent d'autant plus précises que l'arbre grandit. Cette méthode à l'avantage de pouvoir identifier rapidement des séquences d'actions prometteuses dans l'arbre ainsi construit.

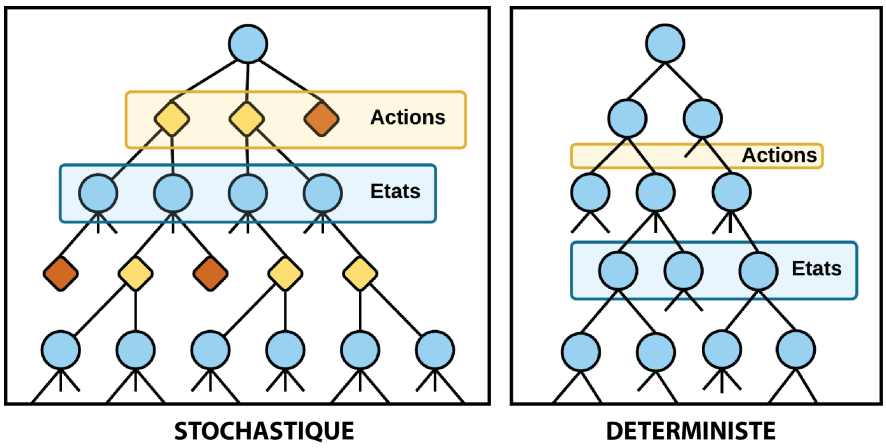


Figure 3.5 **Arbre d'un MDP**. Nœuds états (ronds bleus, nœuds actions (losanges jaunes) et actions terminales (oranges). Les branches partielles correspondent à des sous-arbres non-représentés.

Exploration de l'arbre

Un compteur de visites n est associé à chaque nœud et une estimation de la valeur v à chaque nœud action. Une **exploration de l'arbre** commence à sa racine et est habituellement définie en quatre étapes (Chaslot et al., 2008), illustrées à la figure 3.6 :

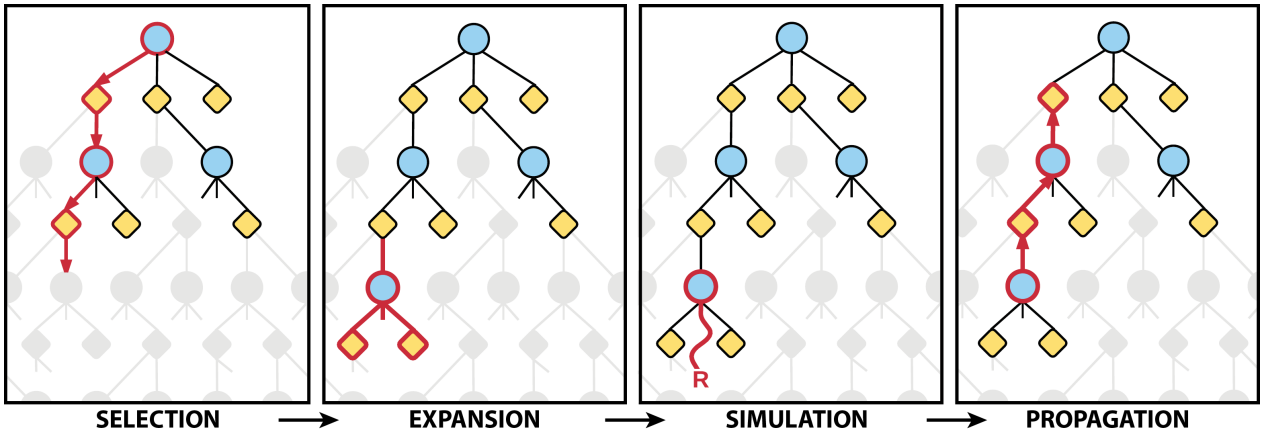


Figure 3.6 **Exploration d'un arbre**. Quatre étapes d'exploration d'un arbre dans la méthode MCTS. Les ronds bleus sont des nœuds états et les losanges jaunes sont des nœuds actions. Les nœuds grisés n'ont pas encore été ajoutés à l'arbre.

1. **Sélection :** On parcourt un chemin à partir de la racine de l'arbre. Le prochain nœud à visiter est déterminé soit par une politique de sélection dans le cas de nœuds actions, soit par échantillonnage de la fonction de transition dans le cas de nœuds états. Cette étape s'arrête lorsqu'on échantillonne un état qui n'a jamais été visitée.
2. **Expansion :** Le nœud état correspondant à ce nouvel état, ainsi que les nœuds actions correspondant aux actions applicables à cet état, sont ajoutés à l'arbre.
3. **Simulation :** Sur base d'une politique par défaut, on effectue une simulation de *rollout* à partir de ce nouvel état. Le résultat de cette simulation peut alors être retourné au nœud parent comme *résultat de l'exploration en aval*.
4. **Rétro-propagation :** En remontant le chemin inverse du nouveau nœud jusqu'à la racine de l'arbre, on additionne progressivement la récompense associée à chaque nœud action visité au résultat de l'exploration, et on met à jour la valeur de ces nœuds d'après ce résultat (équation 3.23).

Nous avons vu plus haut que, dans l'algorithme de *rollout*, une politique de sélection détermine sur quelle action applicable effectuer en priorité une simulation. La méthode MCTS étend ce principe aux nœuds d'un arbre : à chaque nœud état, une politique de sélection détermine le prochain nœud action à visiter. Par analogie, chaque nœud état consiste ainsi en un problème de type *multi-armed bandit* : sélectionner le nœud action à visiter équivaut à tirer le levier d'une machine et les valeurs de ces nœuds actions correspondent aux gains espérées des différentes machines.

Vu sous cet angle, comme illustré à la figure 3.7, une autre façon de présenter une exploration de l'arbre consiste à la définir comme une **exploration récursive de nœuds** :

1. **Exploration d'un nœud état**. S'il s'agit d'une première visite (le nœud vient d'être ajouté à l'arbre), on effectue une simulation de *rollout* à l'aide la politique par défaut et on retourne le résultat de la simulation. Sinon, une politique de sélection détermine une action à appliquer et on retourne le résultat de l'exploration du nœud action correspondant.
2. **Exploration d'un nœud action**. S'il s'agit d'une action terminale, on retourne la récompense terminale. Sinon, on échantillonne l'état suivant depuis la fonction de transition et on explore le nœud état correspondant qu'on initialise si nécessaire (avec ses nœuds actions). On retourne finalement la somme du résultat de cette exploration et de

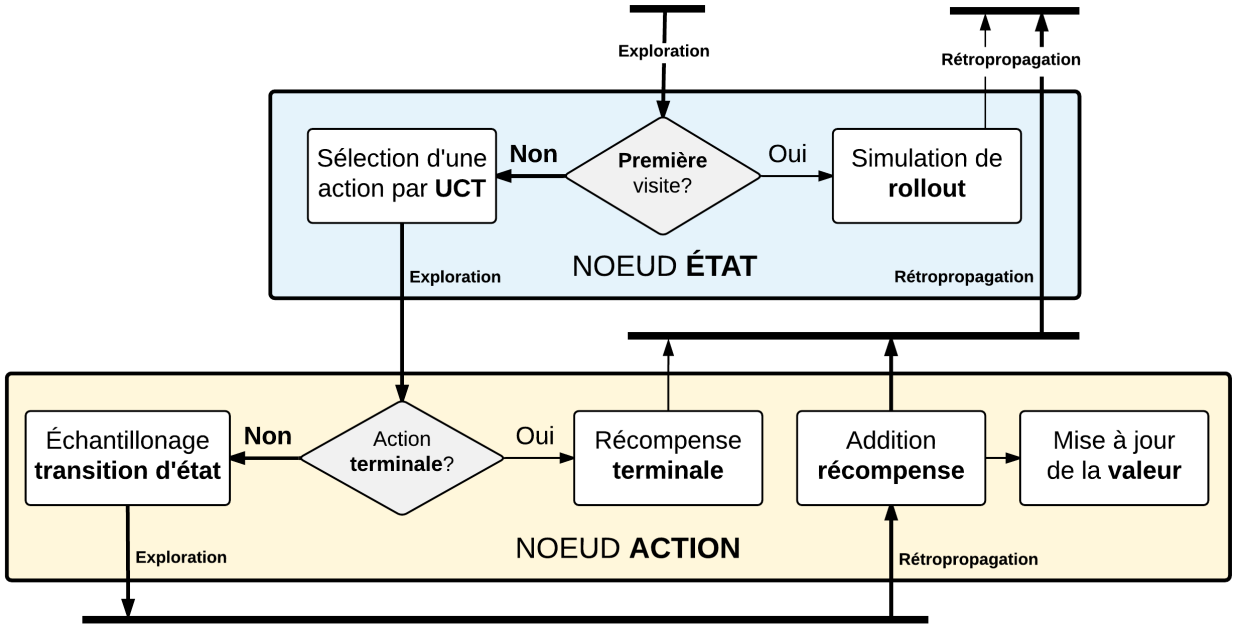


Figure 3.7 **Exploration des nœuds**. Exploration d'un nœud état (en bleu) suivi d'un nœud action (en jaune). Les flèches vers le bas représentent l'exploration d'un nœud enfant, les flèches vers le haut le retour du résultat de l'exploration au nœud parent.

la récompense de l'action considérée, après l'avoir utilisée pour mettre à jour la valeur du nœud (équation 3.23).

Les algorithmes 5 et 6 présentent le pseudo-code pour l'exploration d'un nœud état et d'un nœud action d'après la figure 3.7. Notons qu'il s'agit uniquement d'un exemple d'implémentation et que de nombreuses variations sont possibles sans modifier le principe de fonctionnement général de la méthode.

Algorithme 5 Exploration d'un nœud état

```
1: function EXPLORE-STATE(node)
2:     node.n ← node.n + 1
3:     if node.n = 1 then                                  ▷ Première visite
4:         return ROLLOUT(node.s)                          ▷ Simulation de rollout
5:     else
6:         V ← node.children.v                             ▷ Valeurs des nœuds enfants
7:         N ← node.children.n                             ▷ Visites des nœuds enfants
8:         i ← SELECTION-POLICY(V, N)
9:         return EXPLORE-ACTION(nod.children[i])          ▷ Explorer le ième nœud enfant
10:    end if
11: end function
```

Algorithme 6 Exploration d'un nœud action

```
1: function EXPLORE-ACTION(node)
2:     node.n ← node.n + 1
3:     if TERMINAL(node.a) then                            ▷ Action terminale
4:         return REWARD(node.a)                           ▷ Récompense terminale
5:     else
6:         s ∼ TRANSITION(node.parent.s, node.a)           ▷ Échantillonnage prochain état
7:         i ← find s in node.children.s                   ▷ Nœud enfant correspondant
8:         if i not found then
9:             i ← ADD-CHILD(node, s)                      ▷ Ajout nœud correspondant
10:        end if
11:        r ← EXPLORE-STATE(nod.children[i])              ▷ Explorer le ième nœud enfant
12:        r ← r + REWARD(node.a)                          ▷ Addition de la récompense
13:        node.v ← node.v + (r − node.v) /node.n          ▷ Mettre à jour la valeur
14:        return r
15:    end if
16: end function
```

Itérations, terminaison et action gagnante

Initialement, l'arbre comprend un nœud état comme racine et un ensemble de nœuds actions, qui représentent respectivement l'état initial du système et les actions applicables correspondantes. À chaque exploration de l'arbre, il grandit et les estimations de la valeur v et compteur de visites n des nœuds visités sont mis à jour.

Des explorations de l'arbre continuent à être effectuées jusqu'à ce qu'un **critère de terminaison** soit rempli. On peut simplement se contenter de faire un nombre donné d'explorations. Le critère peut également être basé sur la mémoire ou le temps disponible. Des critères de terminaison plus complexes peuvent également prendre en compte une mesure de la qualité de l'estimation des valeurs. Observons qu'un avantage important de la méthode MCTS est qu'elle peut être interrompue à tout instant ; l'estimation des valeurs sera juste d'autant plus précise que le nombre d'explorations est important.

Une fois les explorations terminées, plusieurs critères existent également pour la **sélection de l'action gagnante** (Chaslot et al., 2007). Deux options couramment utilisées sont :

1. ***Max child*** : Sélection du nœud enfant avec la plus grande valeur estimée v. Il se peut cependant que cette grande valeur estimée ne soit due qu'à un petit nombre de simulations donnant des résultats exceptionnellement élevés par rapport à la valeur réelle de l'état associé.
2. ***Robust child*** : Sélection du nœud enfant avec le plus de visites n. Cette option est plus conservatrice et assure la sélection d'une action prometteuse. Il se peut cependant qu'un autre nœud ait une valeur plus élevée et mériterait d'être exploré d'avantage.

Un compromis pour palier ces inconvénients consiste à utiliser la méthode ***max-robust child*** (Coulom, 2007) qui consiste à continuer à effectuer des explorations jusqu'à ce que le nœud avec la plus grande valeur estimée soit également celui avec le plus grand nombre de visites. L'algorithme 7 présente l'application de la méthode MCTS avec *max-robust child*. Notons que nous imposons également un nombre minimal n_{min} et un nombre maximal n_{max} d'explorations. Après n_{max} explorations, l'algorithme retourne par défaut le résultat avec le plus grand nombre de visites (*robust child*).

Algorithme 7 Monte Carlo Tree Search (avec *max-robust child*)

```
1: function MCTS-Policy(root, n_min, n_max)
2:     repeat
3:         Explore-State(root)
4:         n ← max_i(root.children[i].n)
5:         i_n ← argmax_i(root.children[i].n)          ▷ Le plus de visites
6:         i_v ← argmax_i(root.children[i].v)          ▷ La plus grande valeur
7:     until (root.n > n_max) or (n > n_min and i_n = i_v)
8:     return root.children[i_n].a
9: end function
```

Upper Confidence Bounds for Trees (UCT)

La politique UCB (équation 3.25) considère des distributions de probabilités invariables dans le temps : dans l'exemple du *multi-armed bandit*, si le joueur tire deux fois le bras d'une même machine, son espérance de gain réelle reste identique. Au contraire, dans la méthode MCTS, le résultat de chaque exploration d'un nœud action depuis un nœud état devrait, si tout se passe bien, tendre vers la valeur réelle du nœud action à mesure que le nombre d'explorations augmente. Ces distributions évoluent donc dans le temps.

Kocsis and Szepesvári (2006) étudient l'application de la politique UCB dans le cas de distributions non-stationnaires et en déduisent des conditions nécessaires sur l'évolution temporelle de ces distributions pour assurer la convergence. Ils montrent que l'évolution des distributions dans la méthode MCTS respecte ces conditions pour une politique adaptée, *Upper Confidence Bounds for Trees* (UCT) :

$$j = \underset{i \in [1, n_a]}{\operatorname{argmax}} \left[\bar{v}_i + \kappa \cdot \sqrt{\frac{\ln(n)}{n_i}} \right] \tag{3.26}$$

avec j l'identifiant du prochain nœud action à explorer, n le nombre de visites du nœud état courant, $\bar{v}_i$ la valeur estimée et n_i le nombre de visites du i-ème nœud action. Le paramètre κ permet de favoriser l'exploration ou l'exploitation ; plus il est grand, plus l'influence d'un petit nombre de visites est grande.

Progressive widening

Avec le nombre de visites au dénominateur, la politique UCT va toujours sélectionner en priorité un nœud action qui n'a jamais été visité. Quand l'espace d'action est petit, ce comportement permet d'assurer que chaque action soit visitée au moins une fois. Cependant, lorsque l'espace d'action est grand, le nombre d'explorations nécessaires à cette première visite de chaque action peut nuire fortement à l'efficacité de la méthode MCTS. Comme illustré à la figure 3.8, la croissance en profondeur de l'arbre est interrompue à chaque nœud état : toutes les actions doivent être explorées une première fois avant de pouvoir explorer un état ultérieur.

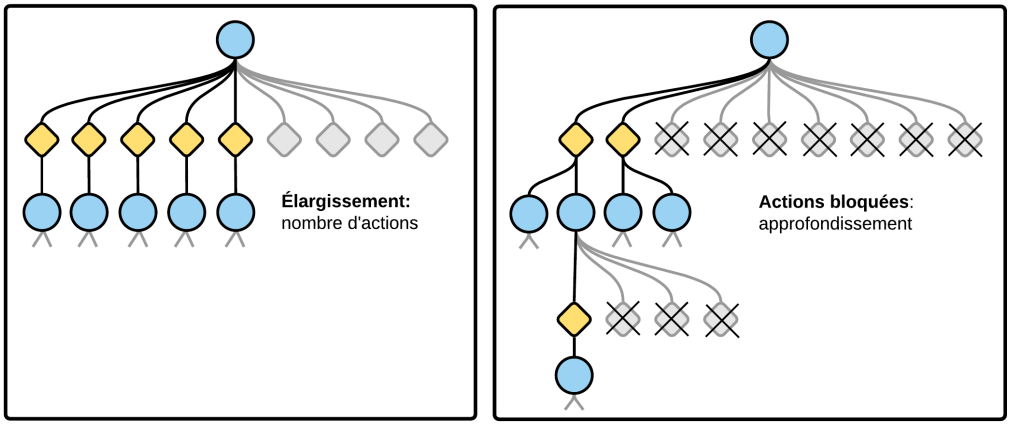


Figure 3.8 **Progressive widening**. En limitant virtuellement le nombre d'actions applicables, on favorise l'accroissement en profondeur de l'arbre. Les nœuds grisés représentent les nœuds actions pas encore explorés et des croix barrent les nœuds actions bloqués par *progressive widening*.

Pour favoriser un accroissement en profondeur de l'arbre, une méthode de ***progressive widening*** (Chaslot et al., 2007; Couëtoux et al., 2011) consiste à adapter la politique de sélection de manière à limiter virtuellement le nombre d'actions applicables à chaque nœud état. Étant donné le nombre de visites n d'un nœud état, le nombre maximum d'actions à considérer est obtenu via une formule empirique de la forme :

$$n_a = \left\lceil \alpha \cdot n^{\beta} \right\rceil$$

avec les constantes $\alpha \in \mathbb{R}^+$ et $\beta \in]0;1]$ sélectionnées pour le problème considéré, notamment en fonction de la dimension de l'espace d'action. Ainsi, le nombre n_a de nœuds actions à

considérer croit progressivement avec le nombre de visites, et la méthode UCT sélectionne le prochain nœud action à visiter dans le sous-ensemble des n_a premiers nœuds actions.

Heuristic progressive widening

Nous proposons une méthode de *progressive widening* alternative qui se base sur une fonction heuristique des actions applicables pour déterminer quelles actions considérer en fonction du nombre total de visites. Définissons $u_i \in \mathbb{R}^+$ comme l'**utilité heuristique** de la i-ème action applicable à partir d'un nœud état donné. On associe ainsi à chaque action une utilité d'autant plus élevée qu'elle semble prometteuse. Avec n le nombre de visites du nœud état, nous considérons pour l'exploration les actions qui remplissent la condition :

$$\frac{u_i}{\max_j(u_j)} \geq \exp(-\alpha \cdot n) \tag{3.27}$$

Initialement, seules les actions les plus utiles sont considérées. À mesure que le nombre de visites augmente, l'exponentielle tend vers zero et des actions de moins en moins utiles sont considérées. La constante $\alpha \in \mathbb{R}^+$ permet de régler le rythme d'ajout de nouvelles actions, et doit être adaptée en fonction des ressources disponibles et du problème étudié.

3.4.4 Application au problème étudié

Dans cette section, nous présentons l'application de la méthode MCTS à notre problème de suivi d'instructions sémantiques par un robot mobile. Commençons par récapituler les principaux éléments nécessaires à l'application de la méthode MCTS :

1. Formulation du problème sous forme de **processus décisionnel de Markov**.
2. Définition d'une **politique par défaut** pour obtenir par simulation de *rollout* une première estimation de la valeur des nœuds actions lorsqu'ils sont ajoutés à l'arbre.
3. Définition d'une fonction d'**utilité heuristique** des actions pour limiter virtuellement par *progressive widening* le nombre d'actions applicables à chaque état durant une exploration de l'arbre (équation 3.27).

Processus Décisionnel de Markov

Comme illustré à la figure 3.9, nous appliquons la méthode MCTS à la planification de haut niveau (section 3.2, page 40) ; d'une part pour sélectionner les objectifs successifs du robot

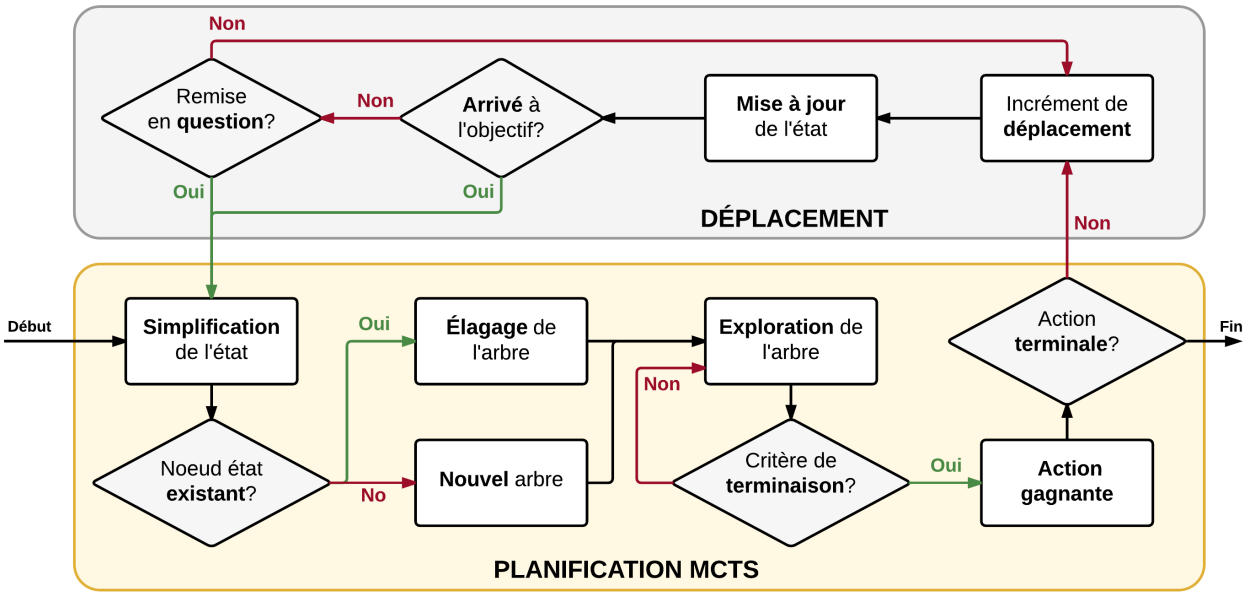


Figure 3.9 **Application de la méthode MCTS**. La méthode MCTS permet la planification d'un nouvel objectif sur base d'un modèle simplifié du système.

et, d'autre part, pour décider de la terminaison de la recherche. Ainsi, il est nécessaire de reformuler le processus décisionnel de Markov présenté à la section 2.4 (page 28) pour prendre en compte ce contrôle à un niveau d'abstraction supérieur.

En particulier, le type d'action *« déplacement élémentaire »* est remplacé par le type d'action *« déplacement vers un objectif »*. Plutôt que la pose adjacente après le prochain déplacement, l'**action** $a(t)$ est alors l'identifiant de la pose navigable sélectionnée comme objectif :

$$\Big(a(t) \in \mathcal{N}_p \Big) \triangleq \begin{array}{l} \textit{Se déplacer vers l'objectif } a(t)\textit{, prendre} \\ \textit{et analyser une photo après chaque dé-} \\ \textit{placement élémentaire et s'arrêter si} \\ \textit{l'objectif doit être remis en question.} \end{array}$$

où l'indice t est ici l'identifiant d'une planification d'objectif : $s(t{+}1)$ est l'état du système à la prochaine planification, quel que soit le nombre de déplacements élémentaires depuis l'état $s(t)$. Notons que l'espace $\mathcal{A}_{s(t)}$ des actions applicables à l'état $s(t)$ est dans ce cas beaucoup plus grand, comprenant l'ensemble des poses navigables de l'environnement plutôt que les seules poses adjacentes.

Si la définition de l'état reste identique, redéfinir les fonctions de transition et de récompense implique de prendre en compte l'ensemble des observations possibles sur le chemin du robot jusqu'à ce qu'il atteigne son objectif ou le remette en question. Maintenant, nous pouvons faire l'hypothèse que l'**objectif n'est jamais remis en question**. Autrement dit, on suppose que la distribution de probabilité sur les cellules des objets, et donc l'état du système, ne peuvent changer de manière significative qu'une fois l'objectif atteint.

La **fonction de transition** reste dans ce cas identique : la pose $p_r(t{+}1)$ du robot est déterministe et correspond à l'objectif $a(t)$ sélectionné, et la distribution de probabilité $\mathbf{b}(t{+}1)$ sur les cellules des objets dépend du résultat d'analyse de la photo prise à l'objectif. Le **coût** (récompense négative) d'une action prend en compte le temps de déplacement par incréments successifs et le temps nécessaire à la prise et l'analyse de photos en chemin :

$$\mathrm{R}\Big(s(t), a(t) \in \mathcal{N}_p\Big) \triangleq -\mathrm{T}_p\Big(p_r(t), a(t)\Big)$$

avec T_p le temps de parcours (section 2.1). Intuitivement, ne pas considérer de remise en question dans le modèle utilisé pour la planification consiste à baser le choix des objectifs successifs uniquement sur ce qu'on pense y observer, sans tenir compte de ce qu'on pourrait observer en chemin. Par analogie, c'est comme si l'étudiant perdu de l'introduction évaluait l'intérêt *d'aller regarder au bout d'un couloir pour voir si les machines distributrices s'y trouvent* sans tenir compte du fait qu'il pourrait éventuellement les repérer avant d'atteindre le bout du couloir.

Politique par défaut et utilité

Nous utilisons la politique heuristique présentée à la section 3.3 (page 42) comme politique par défaut pour les simulations de *rollout*. Nous utilisons également l'utilité globale d'une pose, définie pour cette politique heuristique (équation 3.20), comme fonction d'utilité d'une action de type *« déplacement vers un objectif »* pour l'application de la méthode de *progressive widening* (équation 3.27). Nous associons à l'action terminale (*« arrêter de chercher »*) une utilité unitaire, de sorte à ce qu'elle soit toujours considérée en priorité par la méthode MCTS. Ainsi, dès la première visite d'un nœud, un minimum de deux actions sont considérées : l'action terminale et le meilleur objectif d'après la fonction d'utilité.

Modèle simplifié pour la planification

Précédemment, pour l'estimation d'état, nous avons développé un modèle précis du système avec pour objectif que ce que le robot connaît de l'environnement à un instant donné soit, au possible, complet et proche de la réalité. Pour la planification, une simplification de ce modèle augmente la vitesse des simulations de *rollout* et permet donc une convergence plus rapide de la méthode MCTS vers des actions prometteuses.

Ainsi, nous considérons pour la planification un modèle simplifié du détecteur d'objets, dans lequel on ne tient pas compte des erreurs de localisation (équation 2.15). Nous considérons également un état simplifié en utilisant un nombre n_p réduit de particules pour approximer la distribution de probabilité jointe $\mathbf{b}(t)$ sur les cellules des objets. Au début de la planification d'un objectif, un état simplifié équivalent est obtenu par échantillonnage d'un sous-ensemble des particules de l'état actuel du système (voir figure 3.9).

Comparaison des états

Durant l'exploration de l'arbre, la sélection du prochain nœud état à explorer depuis un nœud action se fait par échantillonnage depuis la fonction de transition. Il s'agit ensuite de comparer l'état ainsi échantillonné avec les nœuds états existants pour sélectionner le nœud à visiter ou, le cas échéant, ajouter un nouveau nœud à l'arbre.

Dans notre cas, à partir d'un état $s(t)$, l'état $s(t{+}1)$ à l'instant suivant est déterminé de manière unique par l'action $a(t)$ et le dernier résultat d'analyse $\mathbf{z}(t{+}1)$ (équation 2.24). Étant donné qu'il est nettement plus facile de comparer deux résultats d'analyse que deux états, nous associons à chaque nœud état le dernier résultat d'analyse par lequel il est déterminé. Durant l'exploration de l'arbre, nous sélectionnons alors le prochain nœud état à visiter par échantillonnage, non plus d'un état, mais bien d'un résultat d'analyse à partir du modèle du détecteur d'objet.

Élagage de l'arbre

Observons qu'il n'est pas nécessaire à chaque planification d'un nouvel objectif de construire et d'explorer un nouvel arbre. Si le nouvel état du système est associé à un nœud qui a déjà été exploré précédemment, il suffit de ne conserver que le sous-arbre dont ce nœud est la racine et de supprimer (*élaguer*) le reste. Ainsi, une partie des explorations réalisées pour la planification des actions précédentes est mise à profit.

CHAPITRE 4 IMPLÉMENTATION ET VALIDATION

Dans ce chapitre, nous validons finalement la méthode développée en présentant nos résultats de simulations. Nous commençons par présenter un exemple d'évaluation des paramètres du modèle de détection d'objet. Nous présentons et commentons ensuite des exemples simulés du déroulement de la recherche des objets successifs par le robot. Nous concluons en évaluant les performances de notre solution et en discutant des résultats obtenus.

4.1 Évaluation des paramètres du détecteur d'objet

Pour le détecteur d'objet, nous considérons que nous disposons d'une caméra *Kinect*[1] qui permet d'obtenir, en plus d'une simple photo, un nuage de points tridimensionnel de ce qui se trouve dans son champ de vision. Nous utilisons l'algorithme DPM (Felzenszwalb et al., 2010), présenté à l'introduction, pour analyser les photos et y détecter des objets. Le cadre retourné par cet algorithme pour chaque objet détecté permet d'en estimer la distance et la direction. Plus précisément, dans le nuage de points, nous conservons le sous-ensemble de points délimité par ce cadre de détection. Nous éliminons ensuite les points en arrière plan par segmentation par rapport à la distance des points (Otsu, 1975). Nous obtenons finalement une direction et une distance approximatives en faisant la moyenne sur les points restants. Des exemples de détections d'objets sont illustrés à la figure 4.1.

Pour évaluer les paramètres de ce détecteur pour une classe d'objet donnée, nous utilisons un modèle de caméra *Kinect* dans l'environnement de simulation *Gazebo*[2]. Nous générons des résultats de détections pour un grand nombre de poses d'un objet de cette classe dans le champ de vison de la caméra. La figure 4.2 montre ainsi quelques exemples de détections de *chaises* dans des photos simulées.

Comme illustré à la figure 4.3, ces résultats permettent d'évaluer la probabilité de détection à distance donnée sur la base du pourcentage de détections positives à cette distance. Le modèle exponentiel de détection (équation 2.10, page 20) est alors adapté à ces probabilités. De même, comme illustré aux figures 4.4 et 4.5, ces résultats permettent d'évaluer l'écart type associé au modèle de localisation (équation 2.12). Observons que, comme nous en avons fait l'hypothèse, l'erreur de localisation est effectivement faible.

1. *Microsoft Kinect 2* : `https://dev.windows.com/en-us/kinect`
2. *Gazebo* (Koenig and Howard, 2004) : `http://gazebosim.org/`

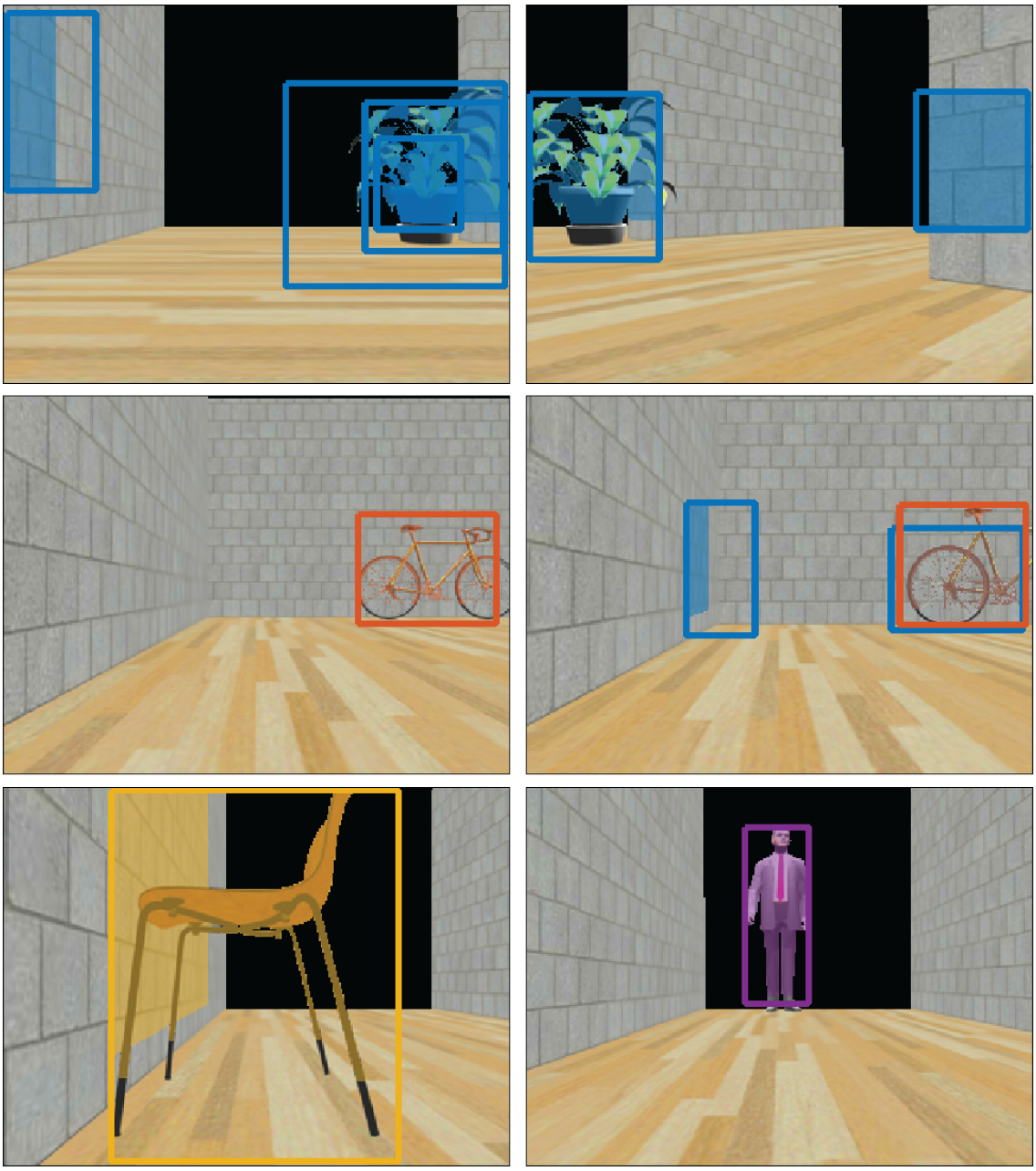


Figure 4.1 **Exemples de détections d'objets** dans un environnement simulé. Les cadres de couleurs représentent les détections retournées par l'algorithme DPM (Felzenszwalb et al., 2010). Les zone colorées représente les points qui, d'après la segmentation, font partie de l'objet détecté.

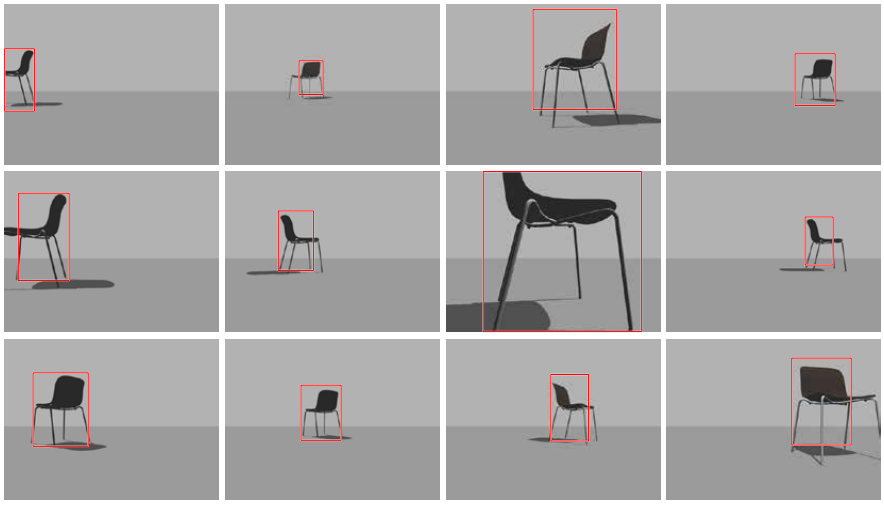


Figure 4.2 **Photos simulées pour évaluer les paramètres**. Exemples de détections dans un ensemble de 2250 photos simulées d'une chaise, dont l'orientation ainsi que la distance et la direction relatives au robot varient.

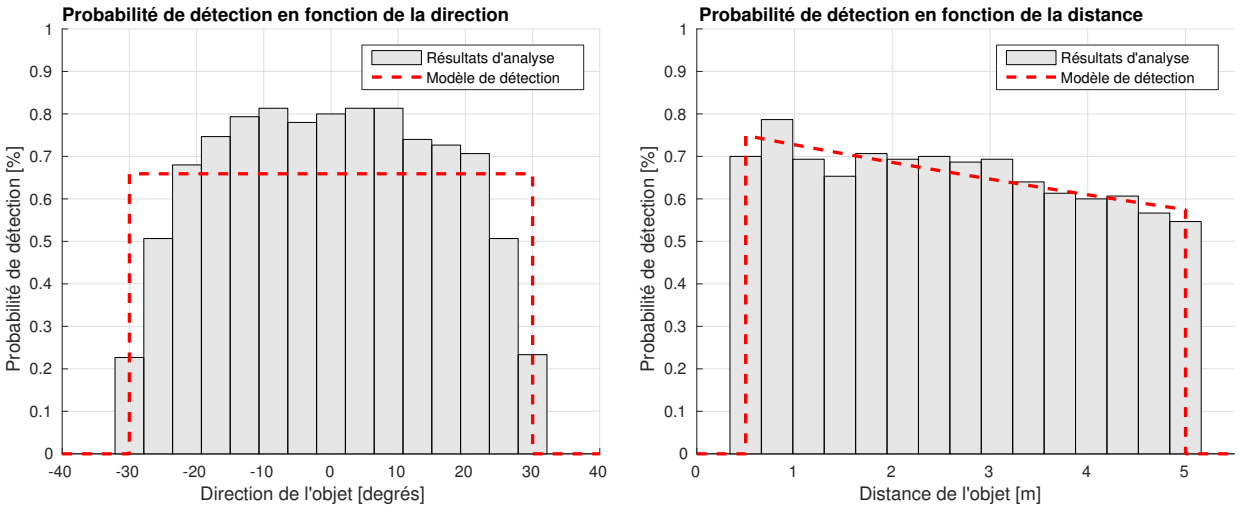


Figure 4.3 **Validation du modèle de détection**. Comparaison du modèle de détection aux données. Observons que nous n'avons pas pris en compte dans notre modèle une diminution significative de la probabilité de détection aux extrémités gauche et droite du champ de vision.

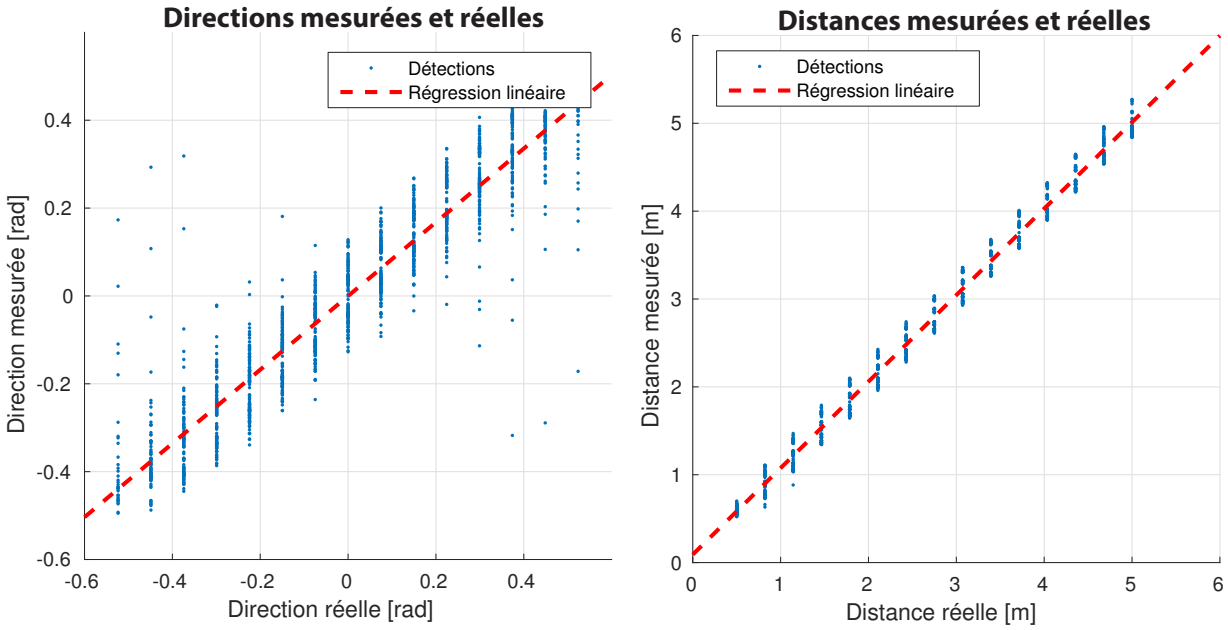


Figure 4.4 **Mesures de distance et de direction**. Comparaison des distances et directions mesurées et réelles de l'objet par rapport à la caméra.

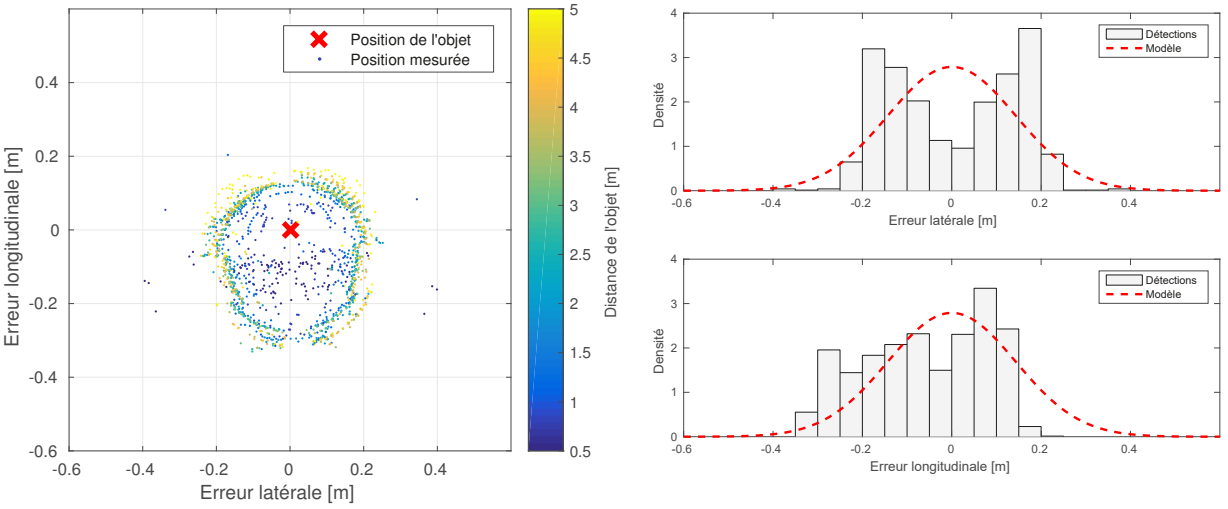


Figure 4.5 **Validation du modèle de localisation**. Les erreurs longitudinales et latérales sont respectivement parallèles et perpendiculaires à l'axe de la caméra. Le creux au centre de la distribution est dû au fait que nous estimons la distance uniquement à partir de la partie visible de l'objet.

4.2 Exemples de déroulement de la recherche

Dans les pages suivantes nous illustrons trois exemples de déroulement de la recherche. D'une part, notre solution, incluant notamment l'estimation d'état, la méthode MCTS et les simulations de *rollout*, est implémentée dans le programme *Matlab*. D'autre part, à défaut d'appliquer notre solution directement sur un robot réel, nous devons simuler le robot, l'environnement et le détecteur d'objet. Une première solution consiste à réaliser ces simulations directement dans *Matlab*, d'après une implémentation des modèles simplifiés présentés au chapitre 2. Pour plus de réalisme, une seconde solution consiste à utiliser l'environnement de simulation *Gazebo* (figure 4.1).

Nous modélisons un robot de type *TurtleBot* équipé d'une caméra *Kinect*. Il a une vitesse linéaire de $0{,}5\,\mathrm{m\,s^{-1}}$ et une vitesse de rotation de $45\,^\circ\,\mathrm{s^{-1}}$. Le champ de vision de la caméra (cellules aux contours grisés dans les illustrations) est limité par un angle de champ d'environ 60° et des distances minimales et maximales de détection respectives de $0{,}5\,\mathrm{m}$ et $5\,\mathrm{m}$ (contraintes par le capteur de distance). Les particularités de chaque exemple sont :

- **Exemple 1 :** En partant de la situation qui nous a servi d'illustration tout au long de ce mémoire (figure 1.3, page 8), les indications de l'opérateur et paramètres du détecteur pour chaque objet sont précisés au tableau 4.1. La simulation est réalisée en entièreté dans *Matlab*, d'après une implémentation des modèles simplifiés présentés au chapitre 2.
- **Exemple 2 :** Même situation que pour le premier exemple (paramètres du tableau 4.1), si ce n'est que la simulation est réalisée à l'aide de l'environnement de simulation *Gazebo*. Nous affichons les détections telles que retournées par l'algorithme DPM.
- **Exemple 3 :** Nous appliquons notre solution à une nouvelle situation, illustrée à la figure 4.6. Les indications de l'opérateur et paramètres du détecteur d'objet pour chaque objet sont donnés au tableau 4.2. Comme pour le premier exemple, la simulation et la planification sont réalisées en entièreté dans le programme *Matlab*.

Dans les illustrations des pages suivantes, les croix de couleurs indiquent les cellules contenant les objets et les cellules colorées représentent les distributions marginales correspondantes. L'arbre construit par la méthode MCTS est projeté sur la carte de l'environnement : les points noirs représentent les objectifs successifs envisagés et les traits noirs les chemins pour atteindre ces objectifs. Le chemin vers l'objectif courant est représenté par des traits verts. La figure 4.7 illustre l'évolution des fonctions d'erreurs et d'utilités des différents objets durant la recherche (voir section 3.3, page 42).

Tableau 4.1 **Instructions et paramètres du détecteur** pour les deux premiers exemples. Situation illustrée à la figure 1.3 (page 8).

objet	n	1	2	3	4
Indication de distance	$\mathbf{d}_n$	10 m	10 m	10 m	15 m
Indication de direction	$\mathbf{a}_n$	90°	0°	−90°	0°
Paramètres de détection	$\pi_{d,n}$	0,8	0,8	0,7	0,8
	$\sigma_{d,n}$	15	15	15	15
Écart-type de localisation	$\sigma_{l,n}$	0,25	0,25	0,25	0,25
Probabilité de parasite	$\pi_{p,n}$	0,005	0,005	0,005	0,005

Tableau 4.2 **Instructions et paramètres du détecteur** pour le troisième exemple. Situation illustrée à la figure 4.6.

objet	n	1	2	3	4
Indication de distance	$\mathbf{d}_n$	15 m	10 m	10 m	5 m
Indication de direction	$\mathbf{a}_n$	0°	90°	180°	−90°
Paramètres de détection	$\pi_{d,n}$	0,8	0,8	0,7	0,8
	$\sigma_{d,n}$	15	15	15	15
Écart-type de localisation	$\sigma_{l,n}$	0,25	0,25	0,25	0,25
Probabilité de parasite	$\pi_{p,n}$	0,005	0,005	0,005	0,005

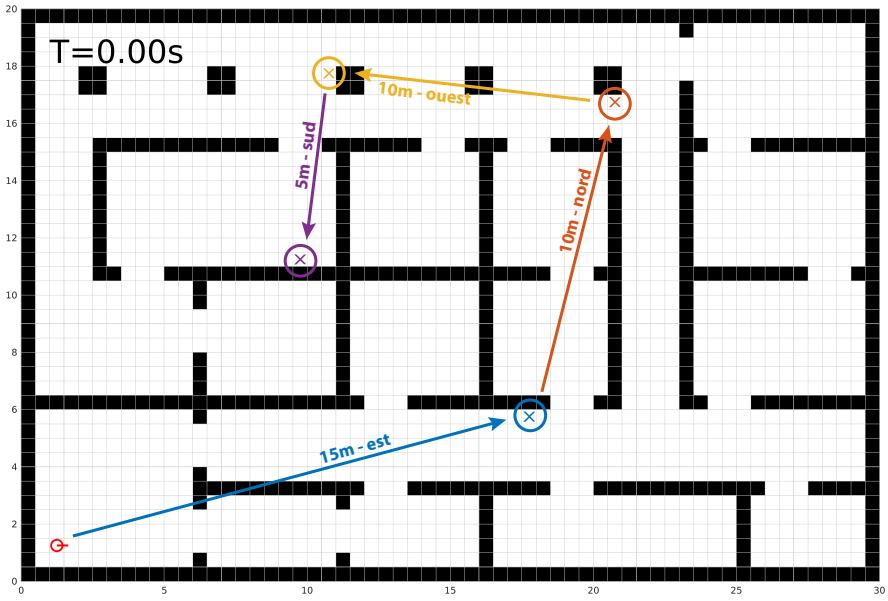


Figure 4.6 **Scénario pour le troisième exemple**

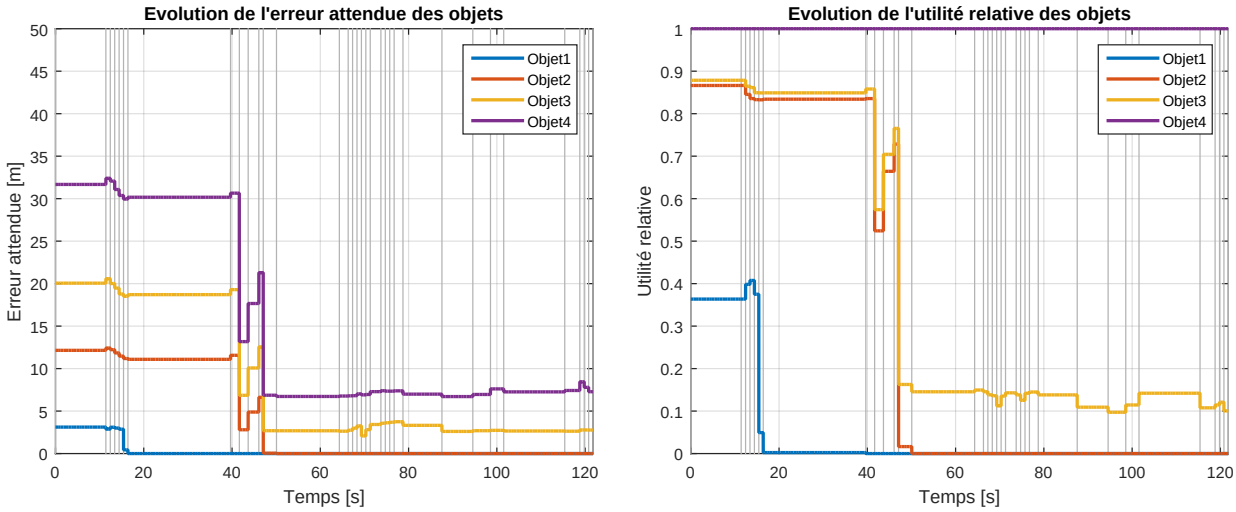
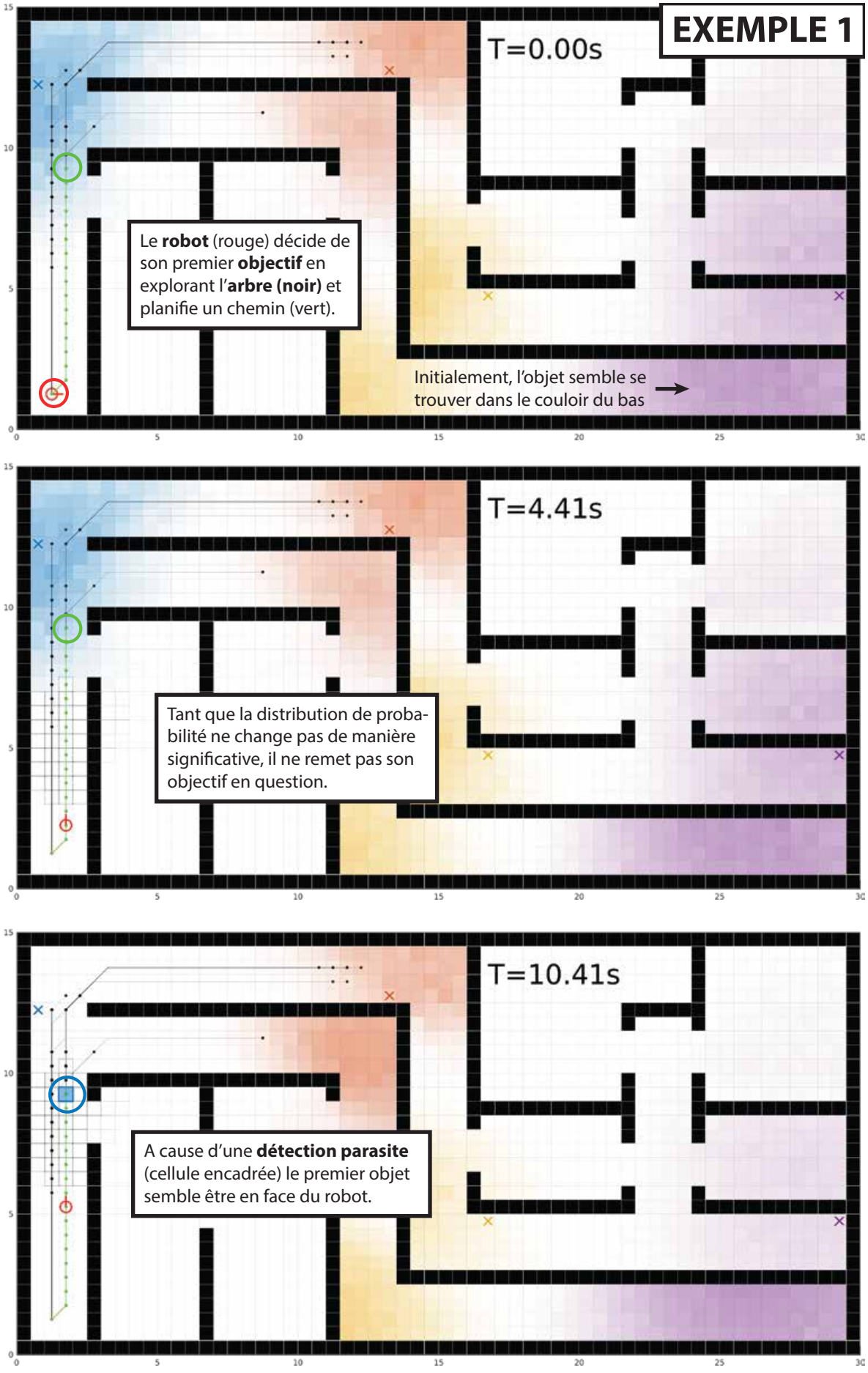
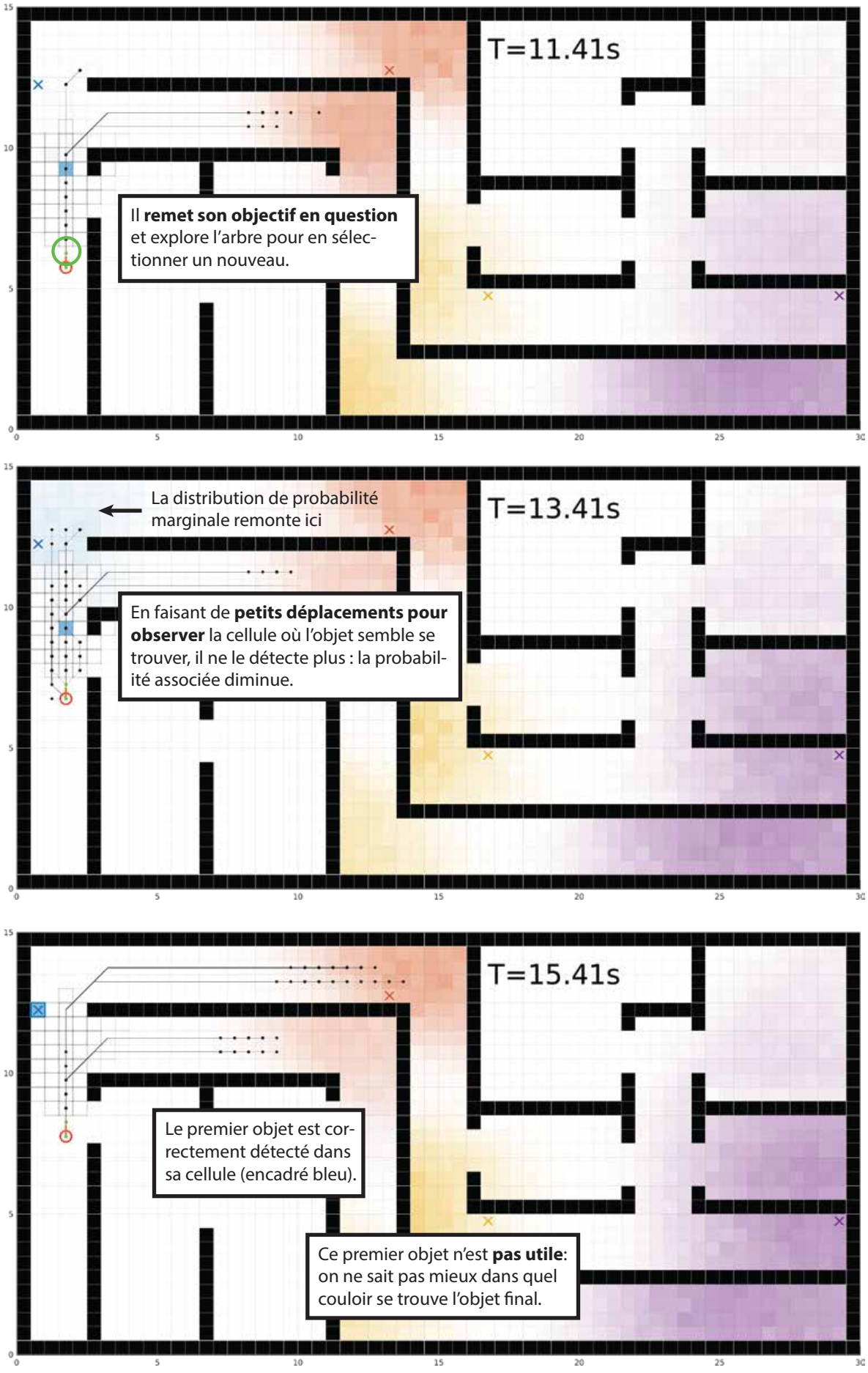
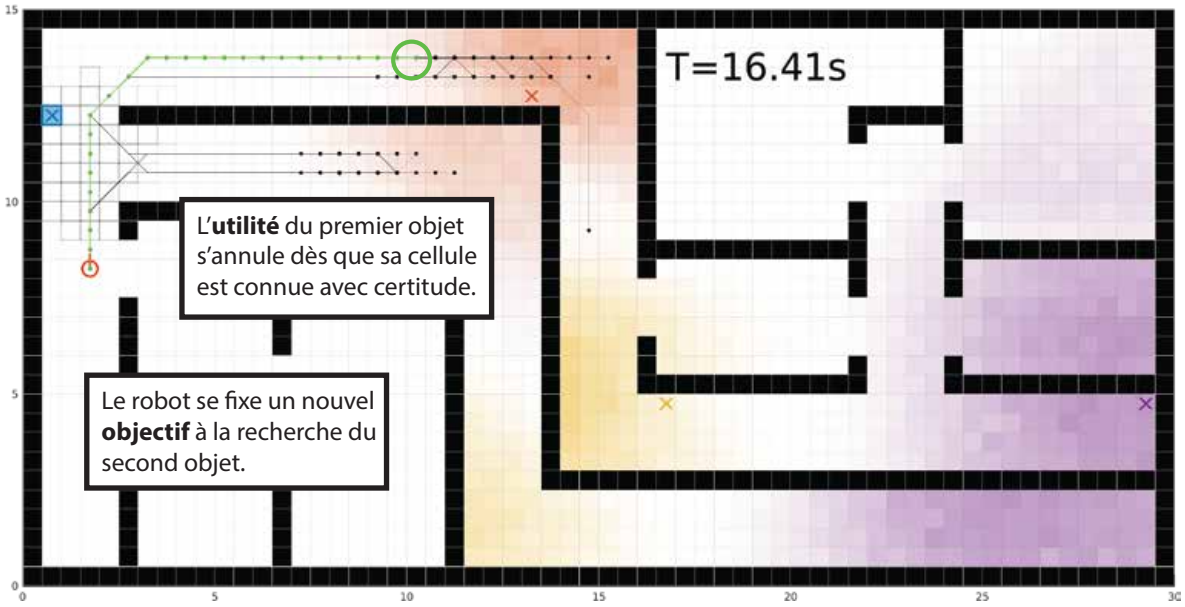
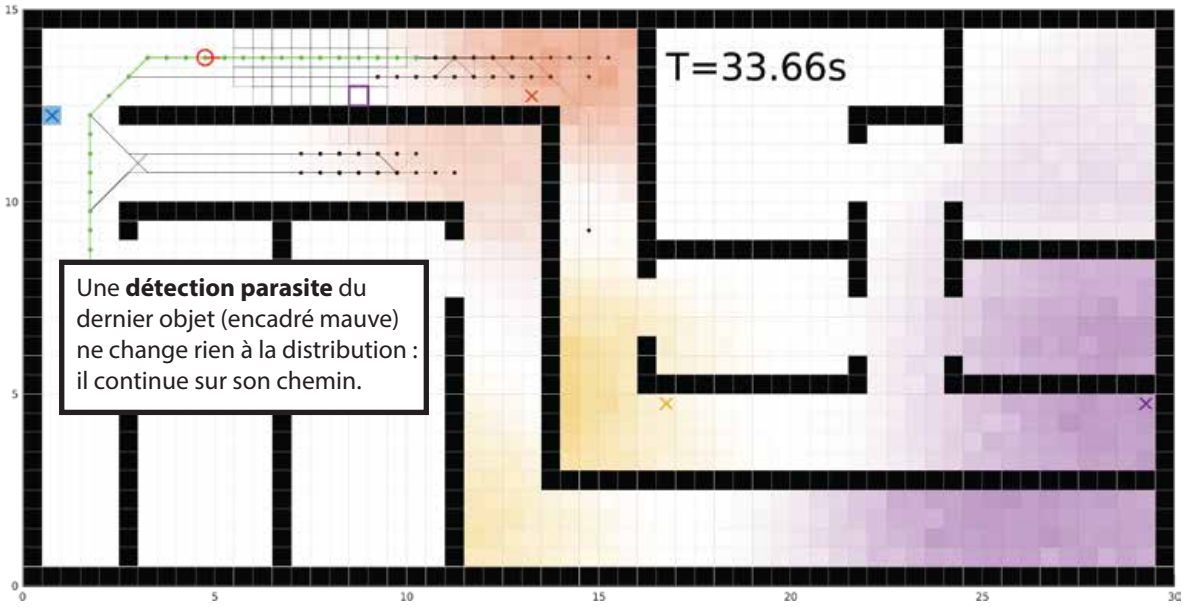


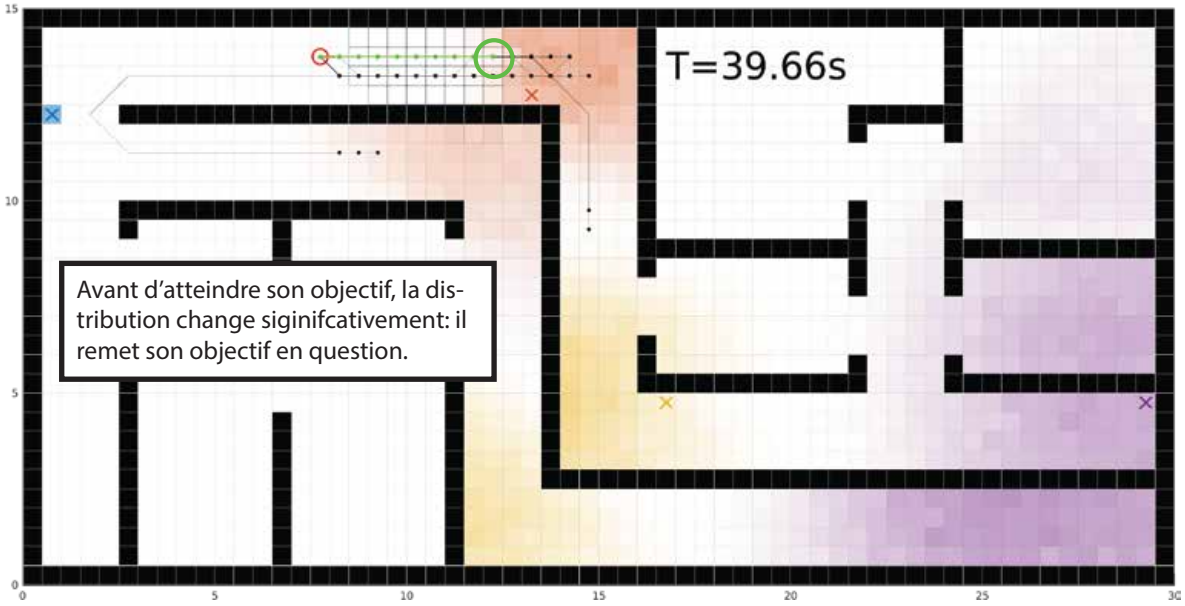
Figure 4.7 **Erreur attendue et utilité relative** (par rapport au dernier objet) pour le premier exemple, mises à jour à chaque nouvelle planification d'un objectif (lignes verticales grises). Observons que l'utilité d'un objet s'annule dès que celui-ci est localisé. Notons également l'utilité importante des deuxième et troisième objets (qui permettent de choisir un des deux couloirs) par rapport au premier.

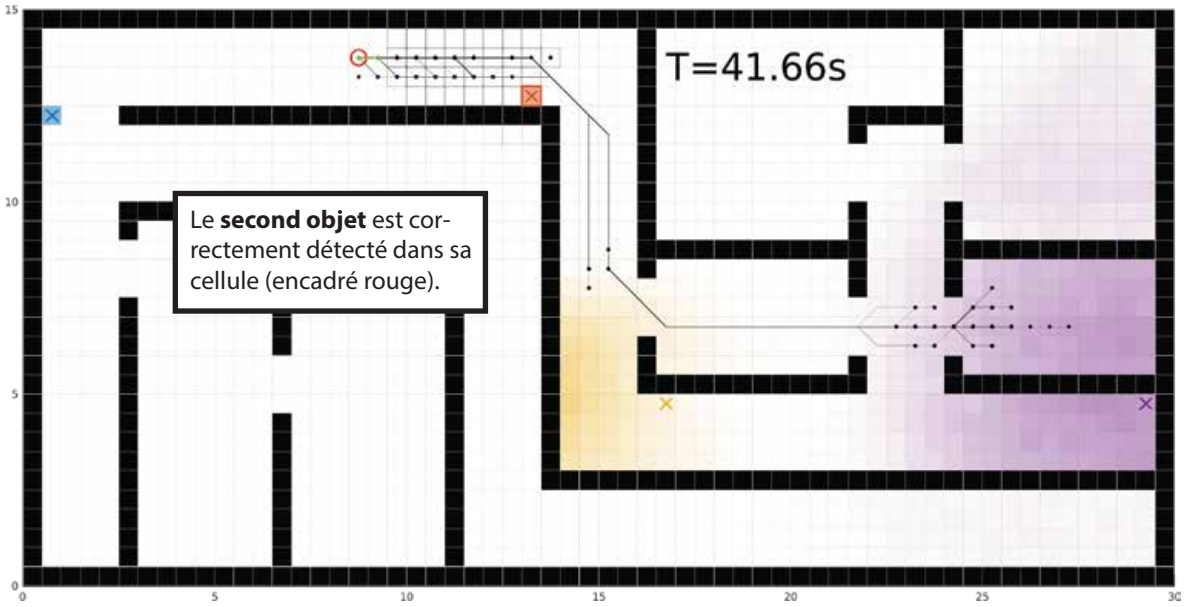

EXEMPLE 1
T=0.00s
Le **robot** (rouge) décide de son premier **objectif** en explorant l'**arbre (noir)** et planifie un chemin (vert).
Initialement, l'objet semble se trouver dans le couloir du bas
T=4.41s
Tant que la distribution de probabilité ne change pas de manière significative, il ne remet pas son objectif en question.
T=10.41s
A cause d'une **détection parasite** (cellule encadrée) le premier objet semble être en face du robot.

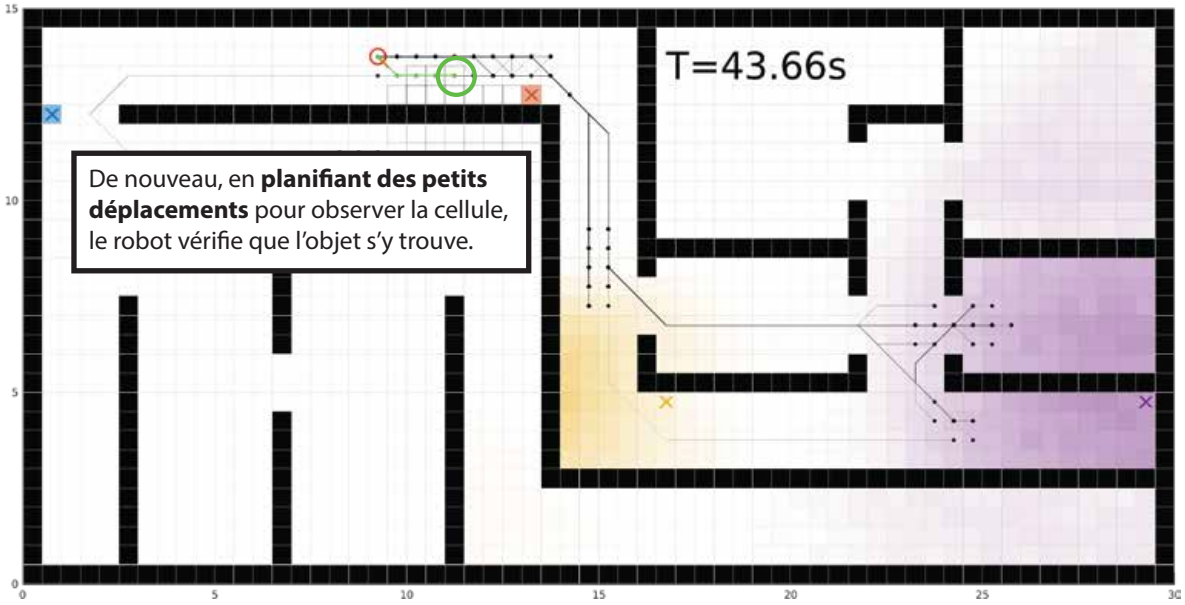

T=11.41s
Il **remet son objectif en question** et explore l'arbre pour en sélectionner un nouveau.
La distribution de probabilité marginale remonte ici
T=13.41s
En faisant de **petits déplacements pour observer** la cellule où l'objet semble se trouver, il ne le détecte plus : la probabilité associée diminue.
T=15.41s
Le premier objet est correctement détecté dans sa cellule (encadré bleu).
Ce premier objet n'est **pas utile**: on ne sait pas mieux dans quel couloir se trouve l'objet final.

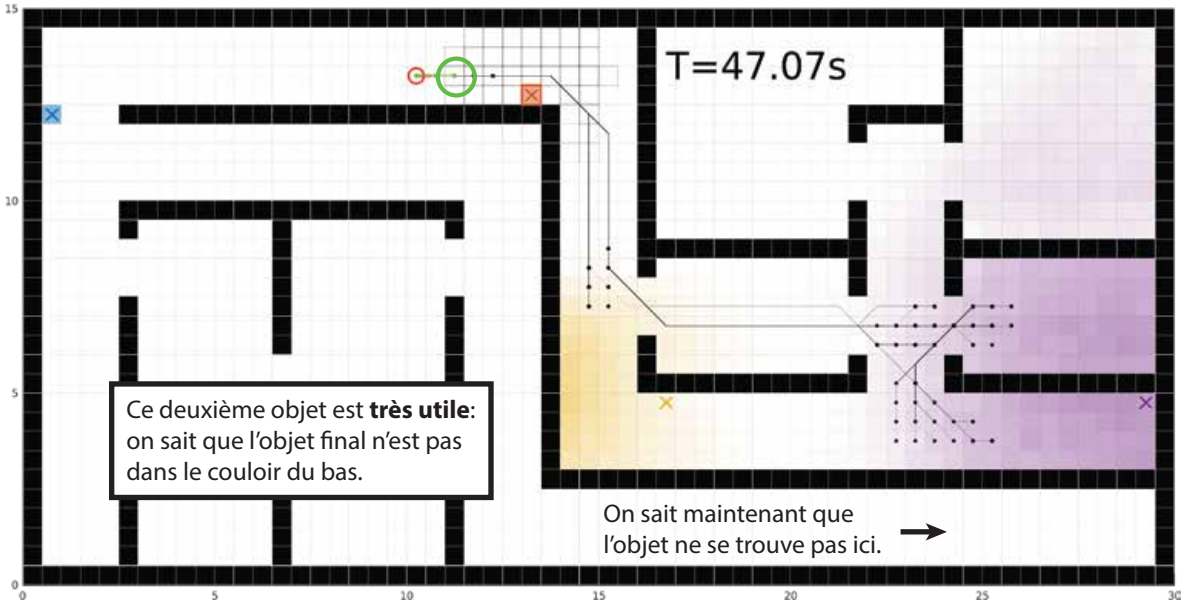

T=16.41s
L'**utilité** du premier objet s'annule dès que sa cellule est connue avec certitude.
Le robot se fixe un nouvel **objectif** à la recherche du second objet.

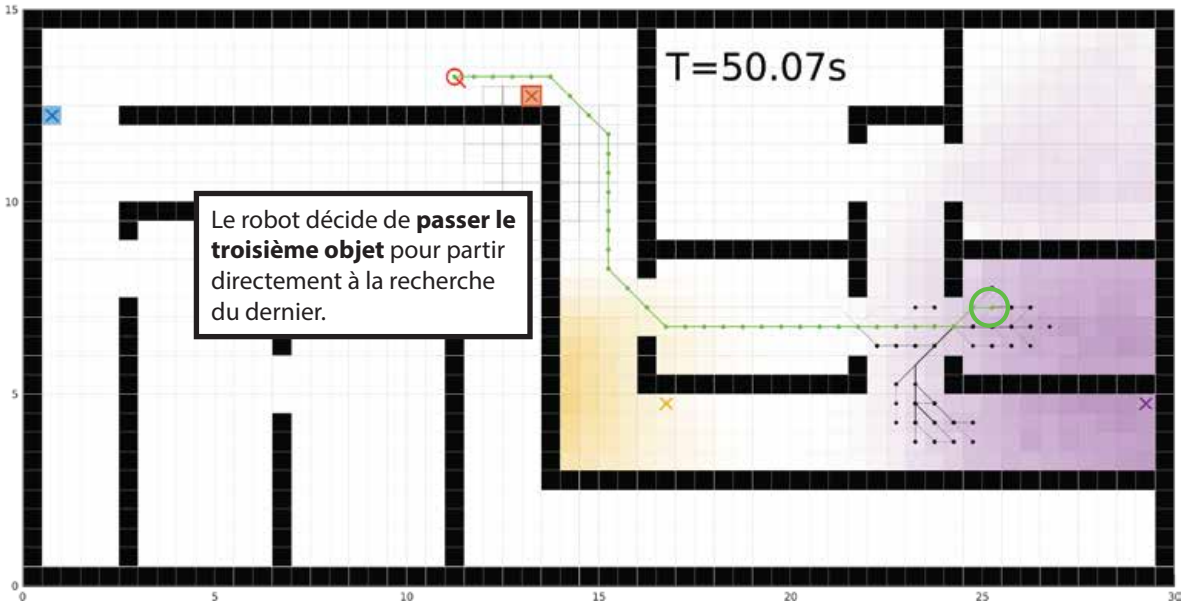

T=33.66s
Une **détection parasite** du dernier objet (encadré mauve) ne change rien à la distribution : il continue sur son chemin.

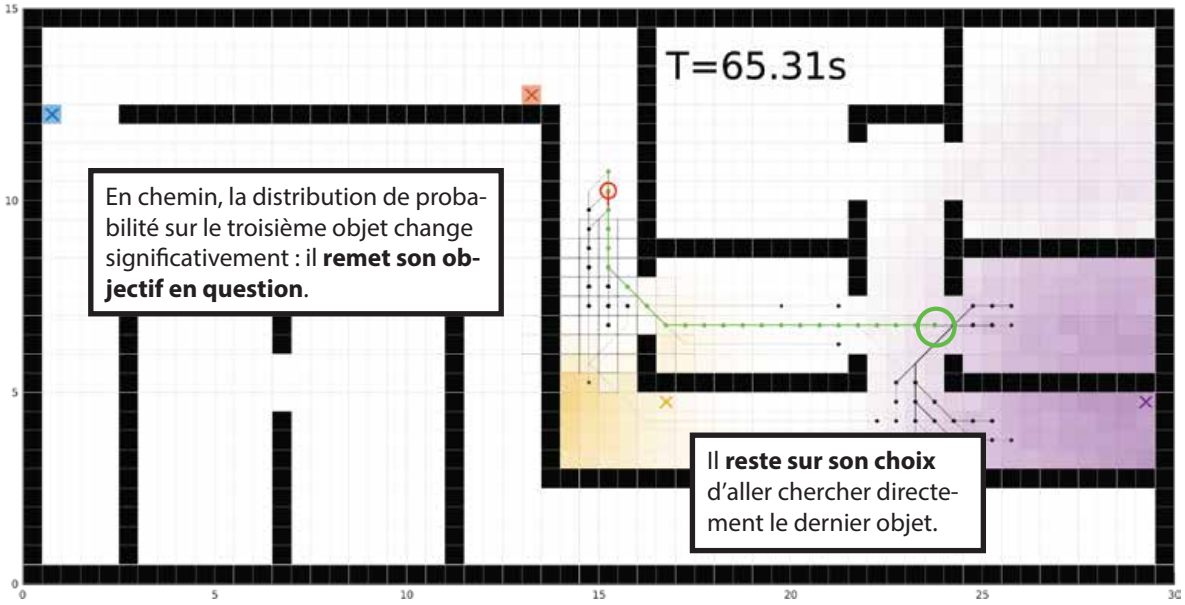

T=39.66s
Avant d'atteindre son objectif, la distribution change siginifcativement: il remet son objectif en question.

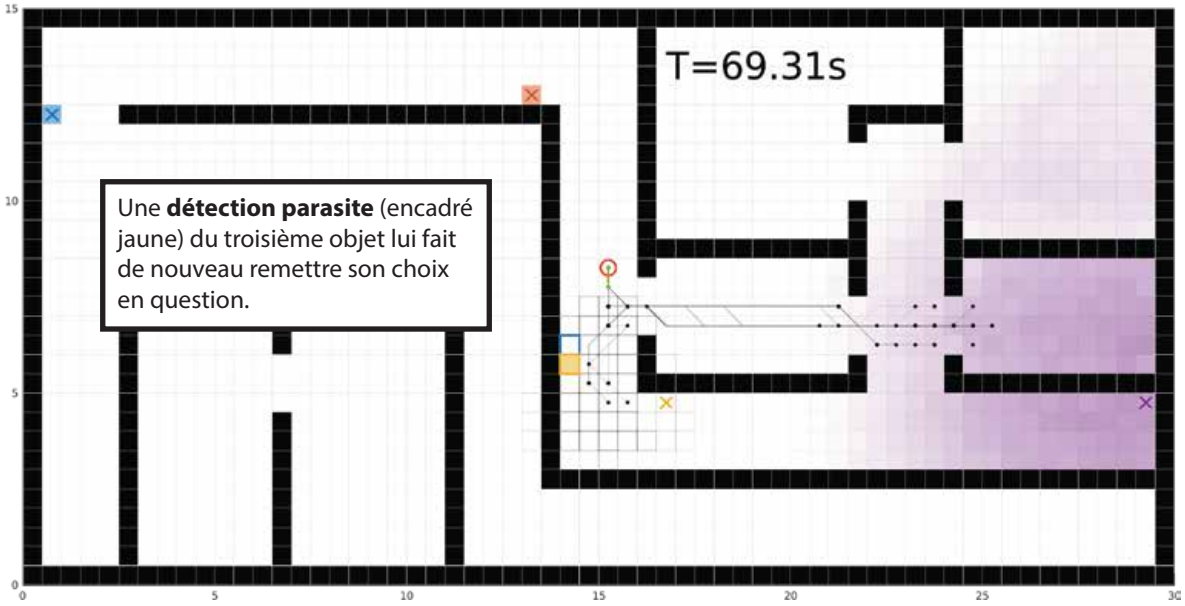

T=41.66s
Le **second objet** est correctement détecté dans sa cellule (encadré rouge).

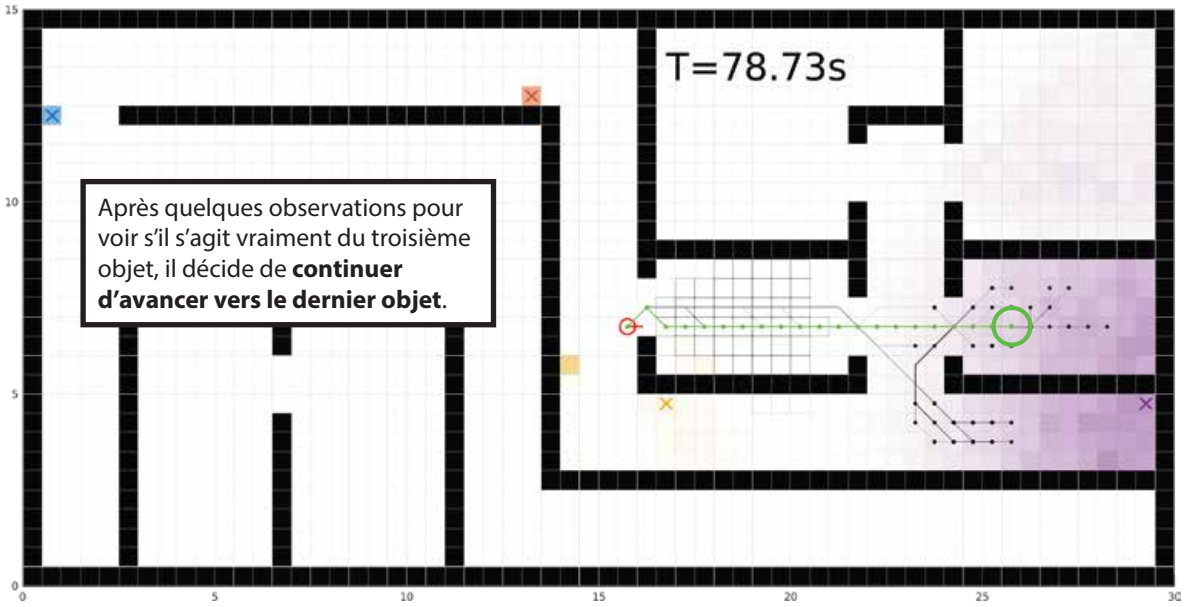

T=43.66s
De nouveau, en **planifiant des petits déplacements** pour observer la cellule, le robot vérifie que l'objet s'y trouve.

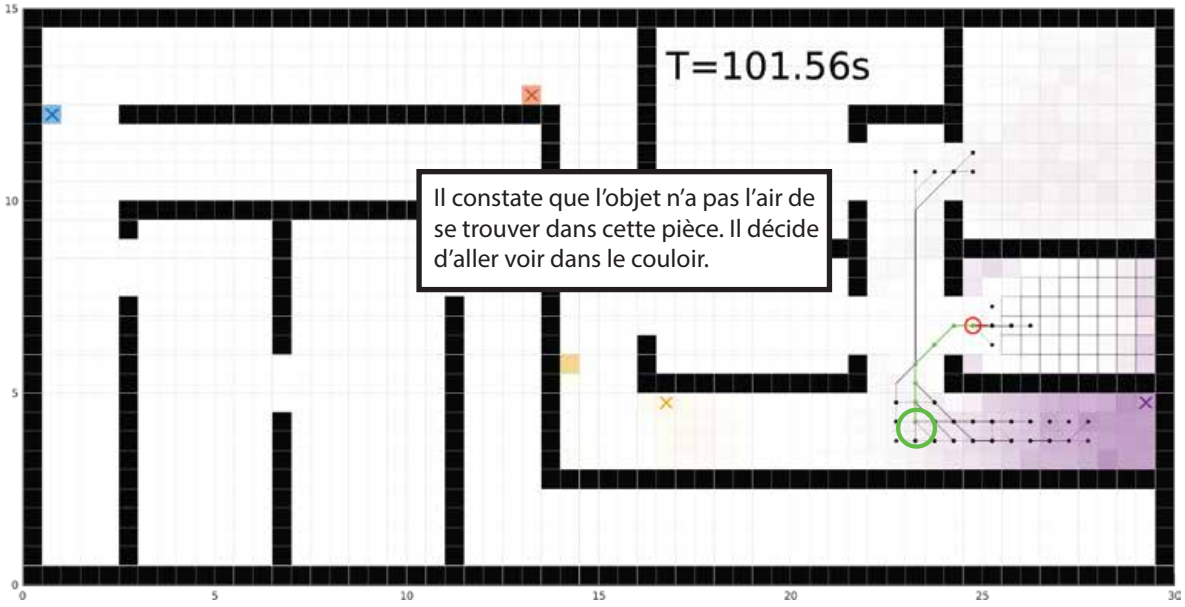

T=47.07s
Ce deuxième objet est **très utile**: on sait que l'objet final n'est pas dans le couloir du bas.
On sait maintenant que l'objet ne se trouve pas ici.

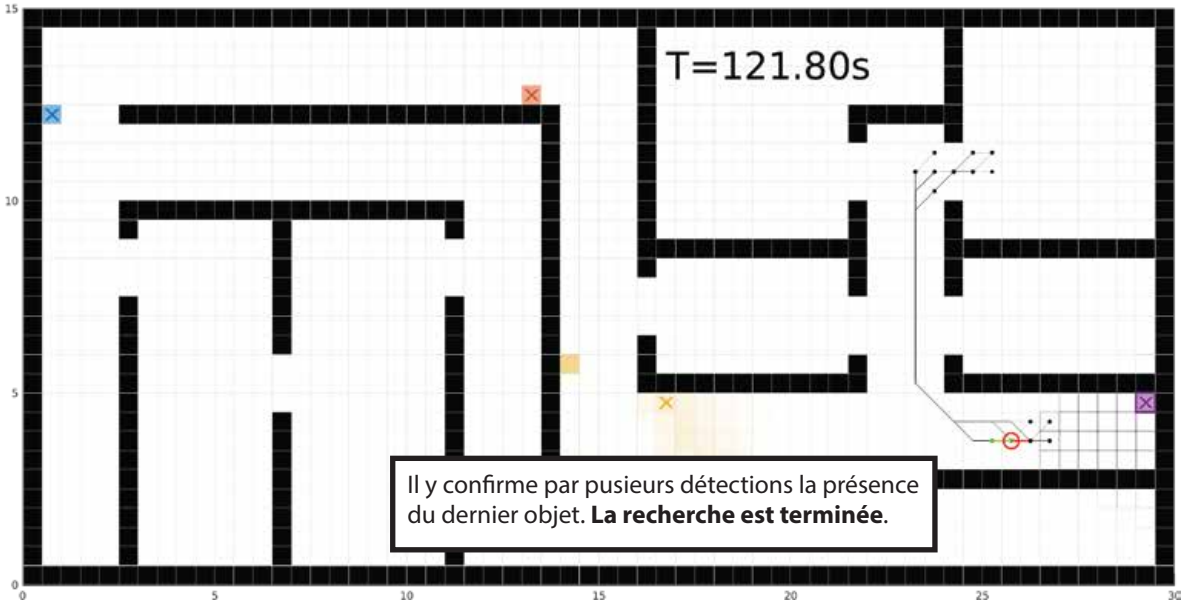

T=50.07s
Le robot décide de **passer le troisième objet** pour partir directement à la recherche du dernier.

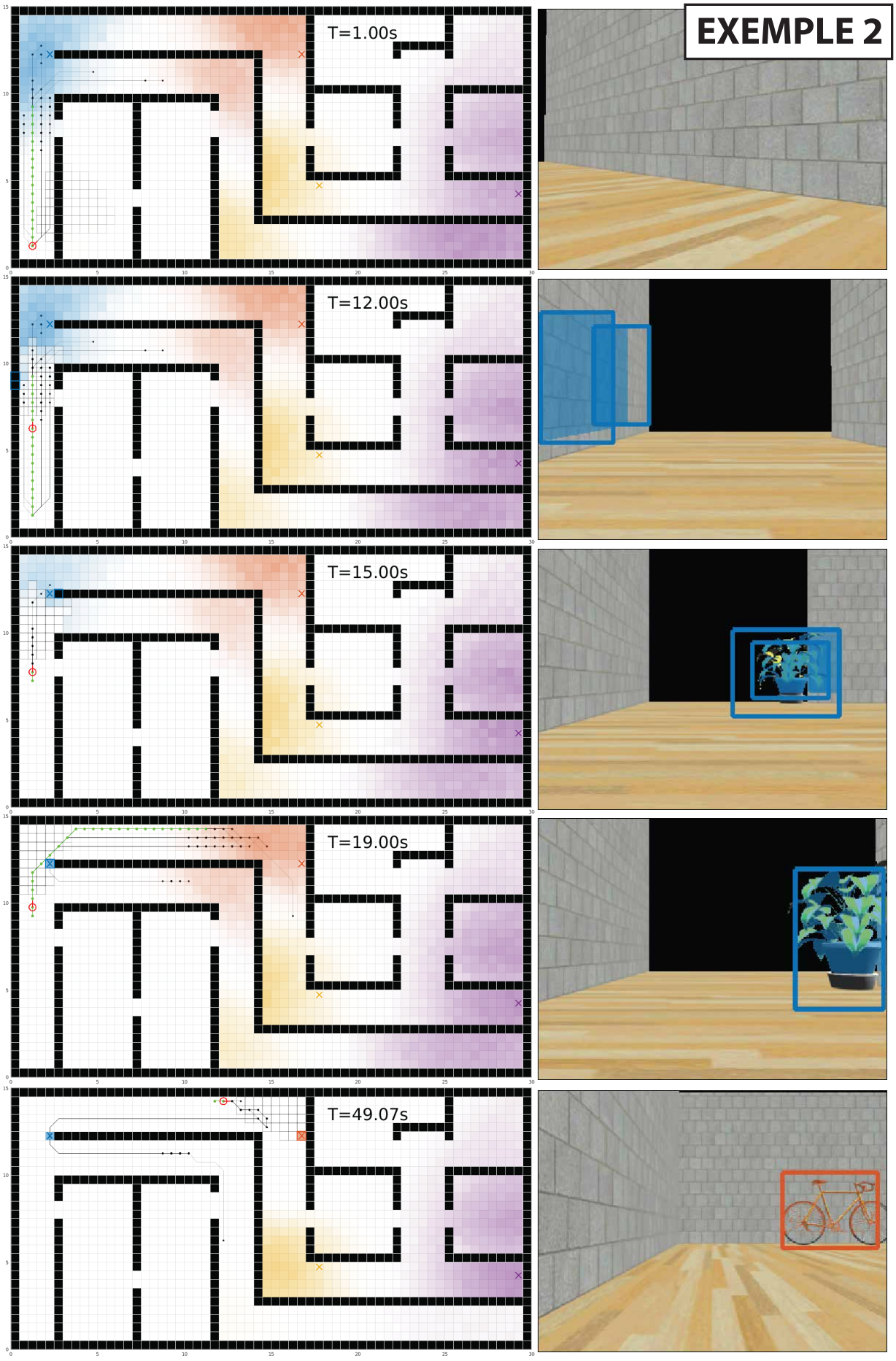

T=65.31s
En chemin, la distribution de probabilité sur le troisième objet change significativement : il **remet son objectif en question**.
Il **reste sur son choix** d'aller chercher directement le dernier objet.

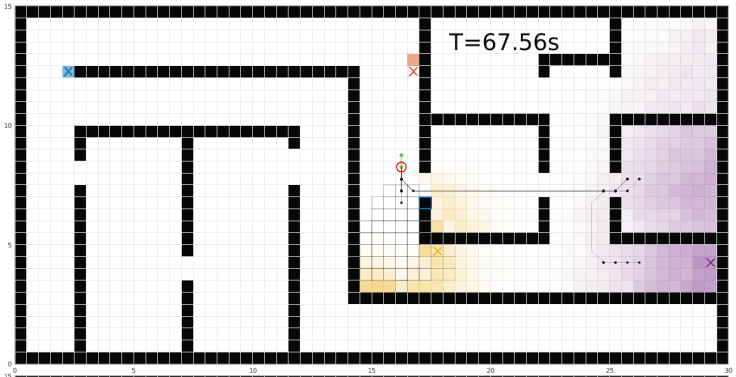

T=69.31s
Une **détection parasite** (encadré jaune) du troisième objet lui fait de nouveau remettre son choix en question.

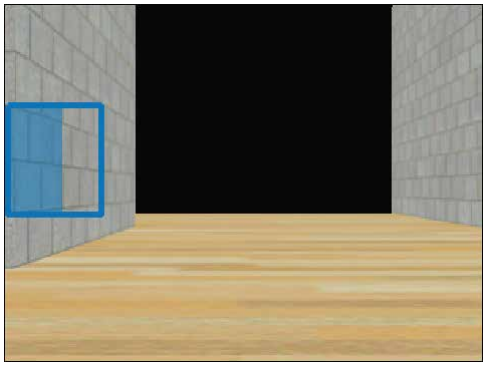

T=78.73s
Après quelques observations pour voir s'il s'agit vraiment du troisième objet, il décide de **continuer d'avancer vers le dernier objet**.

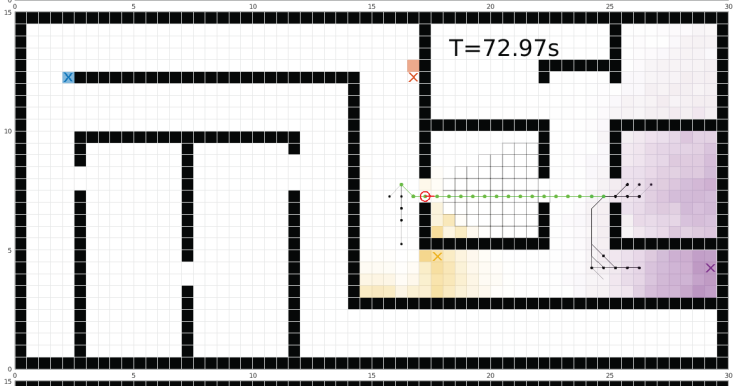

T=101.56s
Il constate que l'objet n'a pas l'air de se trouver dans cette pièce. Il décide d'aller voir dans le couloir.

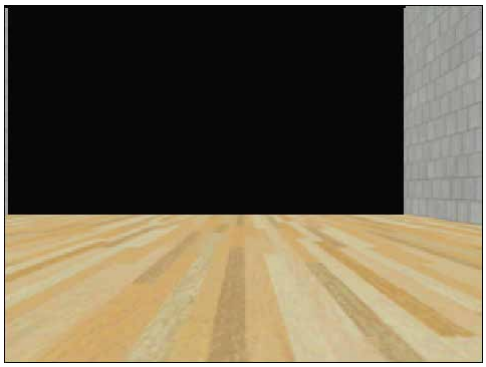

T=121.80s
Il y confirme par pusieurs détections la présence du dernier objet. **La recherche est terminée**.

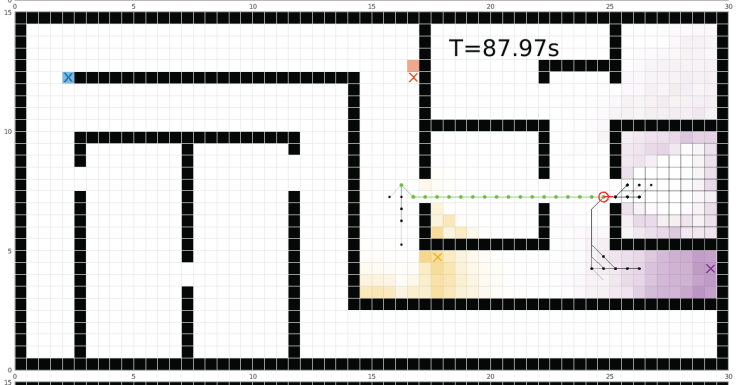

EXEMPLE 2
T=1.00s
T=12.00s
T=15.00s
T=19.00s
T=49.07s

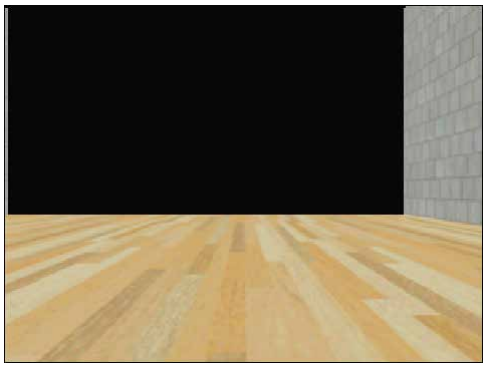

T=67.56s

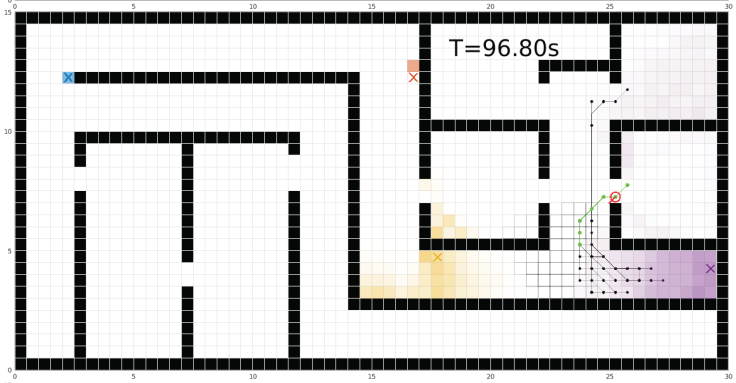


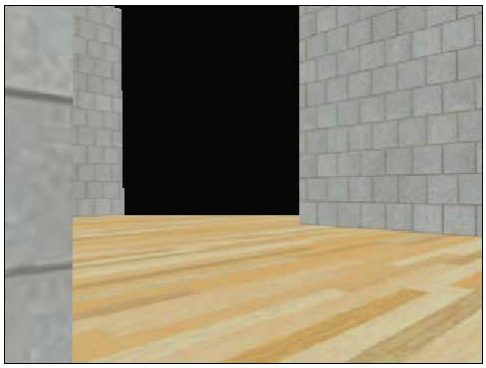

T=72.97s

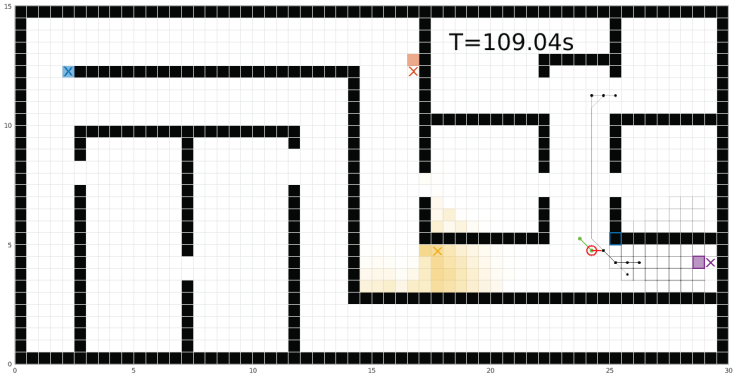


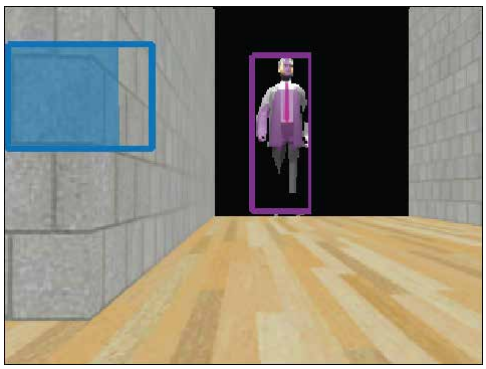

T=87.97s

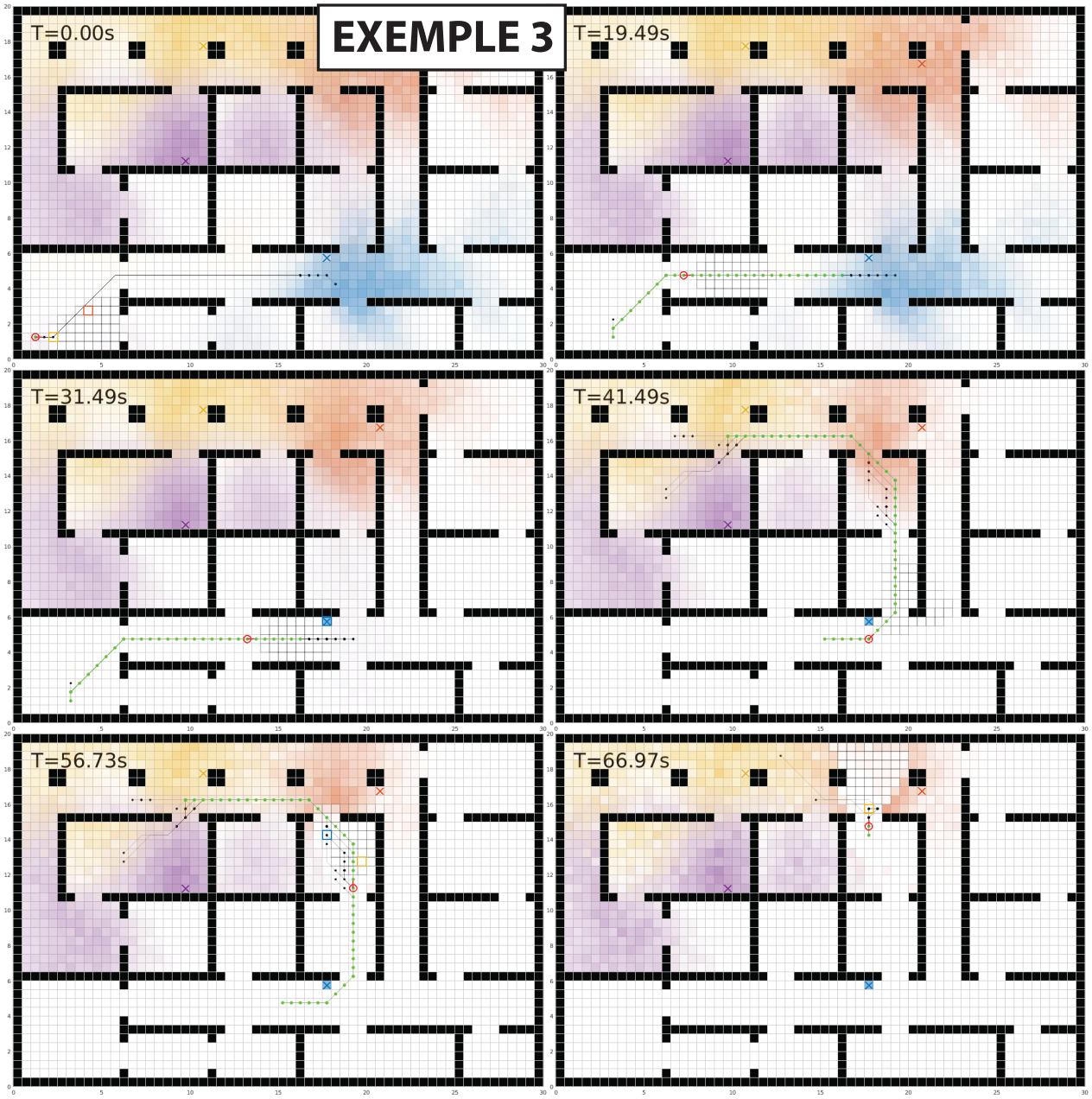


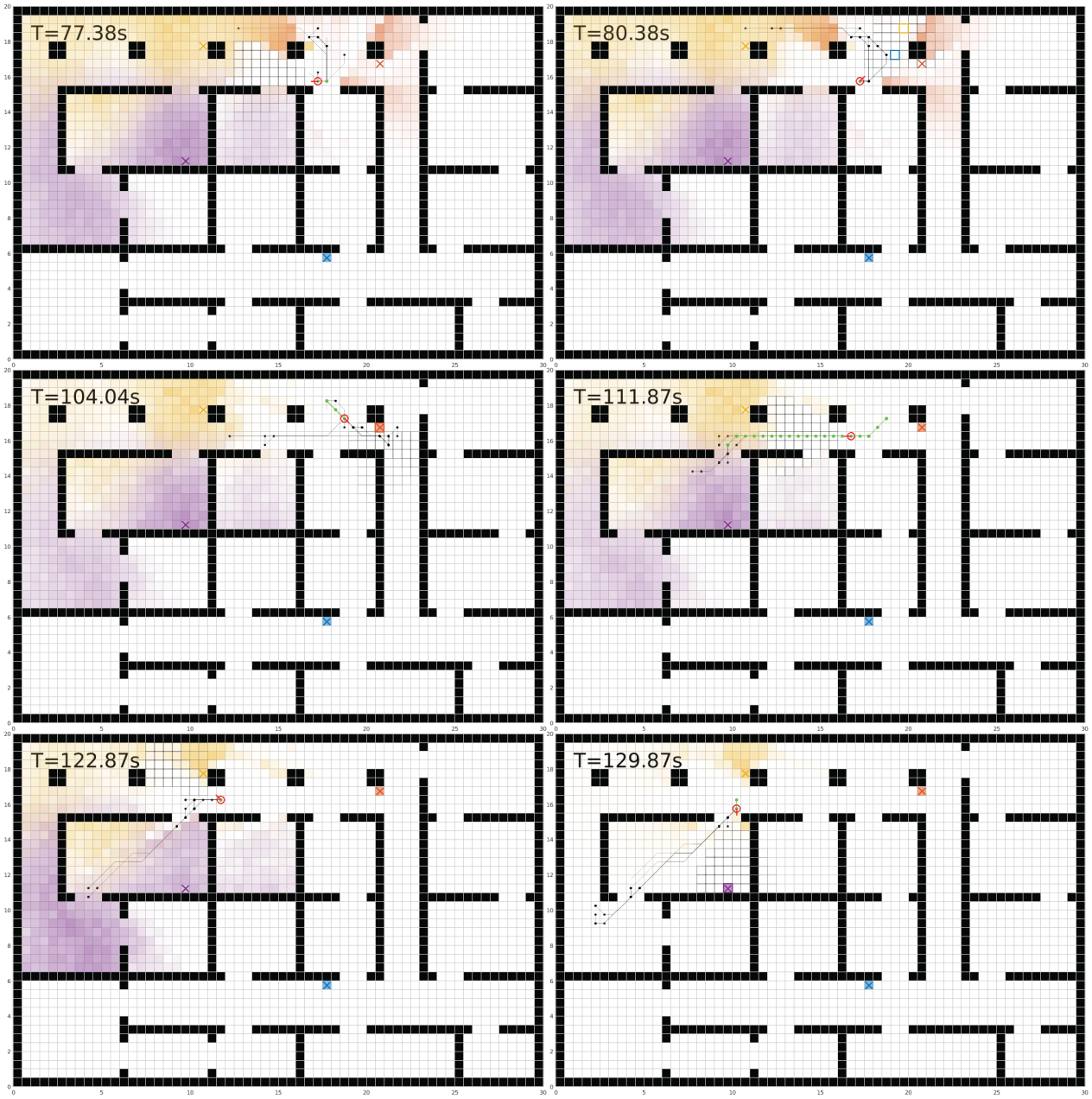

T=96.80s




T=109.04s




EXEMPLE 3
T=0.00s
T=19.49s
T=31.49s
T=41.49s
T=56.73s
T=66.97s


T=77.38s
T=80.38s
T=104.04s
T=111.87s
T=122.87s
T=129.87s

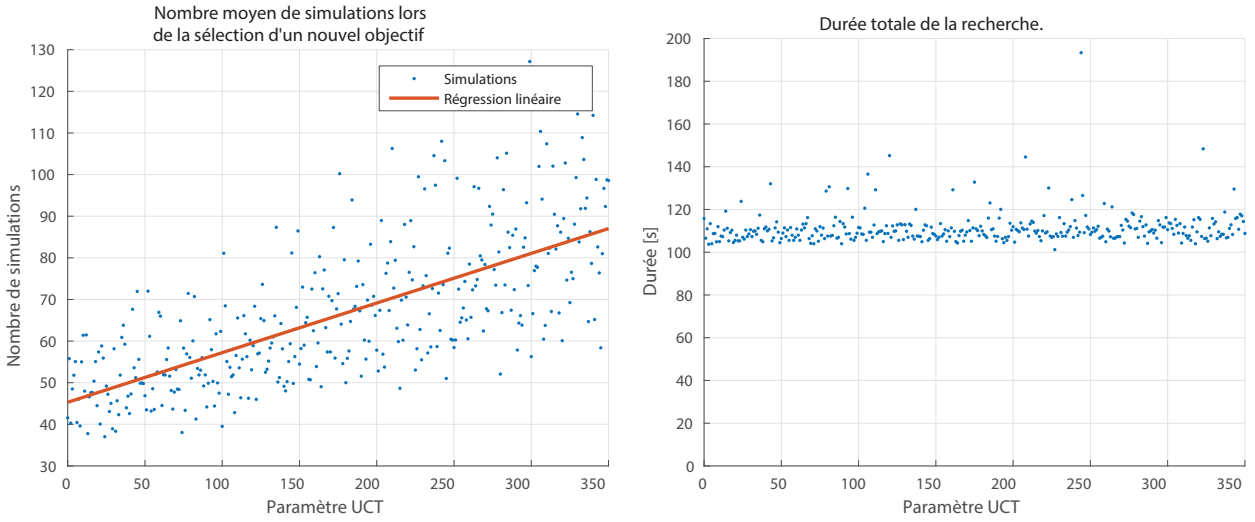


Figure 4.8 **Influence du paramètre de la méthode UCT**. Le nombre moyen de simulations lors de chaque planification d'un nouvel objectif augmente effectivement avec ce paramètre. Nous n'observons pas de différence significative de performance (durée de la recherche).

4.3 Influence du paramètre de la méthode UCT

Comme nous l'avons présenté à la section 3.4.3 (page 57), la méthode UCT sert de politique de sélection lors de l'exploration de l'arbre dans la méthode MCTS et permet de faire un compromis entre l'exploitation des nœuds actions les plus prometteurs et l'exploration des nœuds actions peu visités. Un unique paramètre κ permet de régler ce comportement et de favoriser soit l'exploitation, soit l'exploration.

Pour évaluer l'influence de ce paramètre sur les performances de la méthode MCTS, nous avons réalisé 350 simulations du système avec un scénario identique au premier exemple ci-dessus pour différentes valeurs de κ. Par souci de performance, ces simulations ont été réalisées entièrement dans *Matlab* plutôt qu'avec l'aide de *Gazebo*. La figure 4.8 illustre les résultats de ces simulations. Comme attendu, le nombre moyen de simulations lors de chaque planification d'un nouvel objectif augmente effectivement avec le paramètre κ, d'avantages d'actions moins prometteuses étant explorées (voir *max-robust child*, page 56).

4.4 Évaluation et discussion des résultats

Afin d'évaluer les performances de la méthode MCTS, nous avons réalisé un total de 600 simulations du système avec un scénario identique au premier exemple ci-dessus. Pour une moitié de ces simulations, les objectifs du robot étaient planifiés par la politique heuristique, pour l'autre moitié, par la politique MCTS. Par souci de performance, ces simulations ont été réalisées entièrement dans *Matlab*. La figure 4.9 illustre l'évolution moyenne de l'erreur attendue (équation 3.17, page 42) sur le dernier objet suivant la politique utilisée. Par analogie à la figure 4.7, cette moyenne correspond à la courbe de l'erreur attendue pour le quatrième objet ; notons la chute significative suite à la détection de chaque objet intermédiaire.

Malheureusement, nous n'observons pas l'amélioration significative des performances espérée avec la méthode MCTS. Dans ce cas, même si la différence de performance entre les deux politiques n'est pas significative, le dernier objet semble être localisé plus rapidement avec la politique heuristique qu'avec la méthode MCTS. Une raison qui peut expliquer ces faibles performances de la méthode MCTS est une croissance en profondeur de l'arbre insuffisante. Avec un grand nombre d'objectifs applicables à chaque état et un grand nombre de résultats d'analyses possibles après chaque action, de nombreuses exploration de l'arbre résultent en la création rapide d'une nouvelle branche plutôt qu'en l'exploration d'une branche existante.

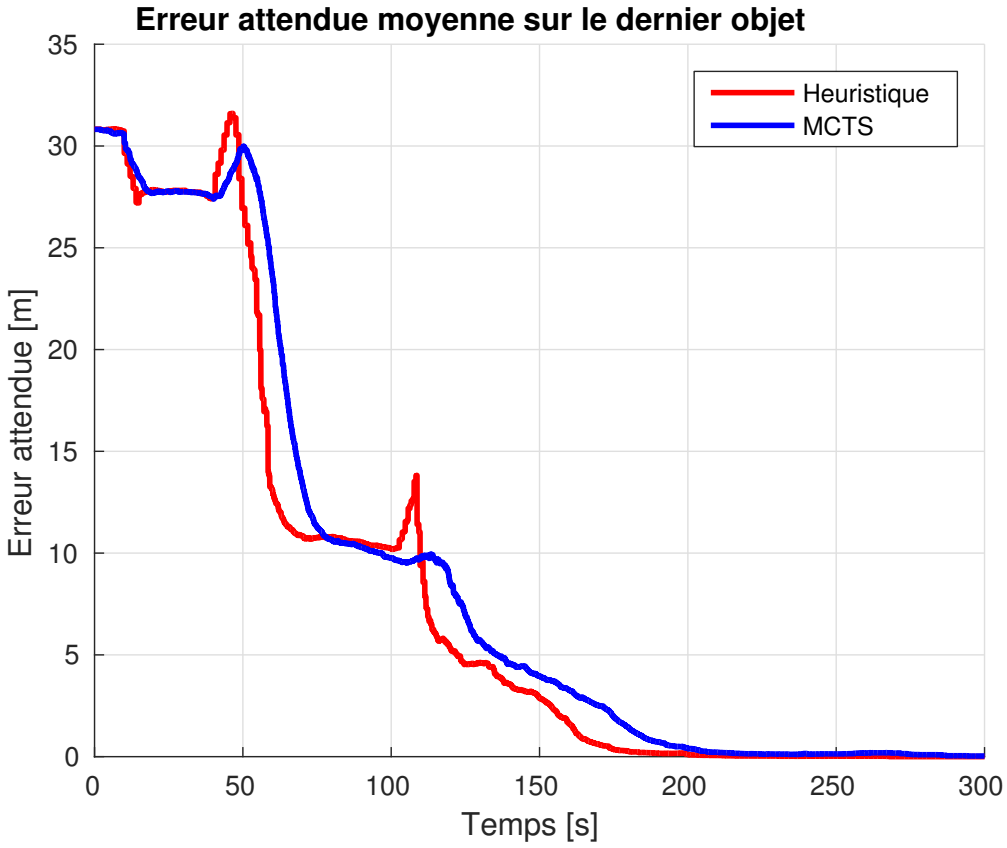


Figure 4.9 **Comparaison des résultats de simulation**. Comparaison de l'évolution moyenne de l'erreur attendue sur le dernier objet entre la politique heuristique et la politique MCTS. Le dernier objet est localisé parfaitement lorsque l'erreur attendue est nulle ; la politique heuristique semble ainsi être plus rapide.

CHAPITRE 5 CONCLUSION

Dans ce mémoire de maîtrise, nous avons présenté la formulation et la résolution d'une version simplifiée du problème de suivi d'instructions sémantiques pour un robot mobile. Nous passons rapidement en revue les apports de ce travail. Nous concluons en présentant les limitations de notre solution et des pistes d'améliorations.

5.1 Synthèse des travaux

Notre version du problème de suivi d'instructions sémantiques repose sur une série d'objets identifiables par le robot et une série d'instructions qui spécifient approximativement la position relative de chaque objet par rapport au précédent. En faisant l'hypothèse d'un environnement connu et d'une localisation parfaite, ce problème est fortement simplifié tout en conservant la problématique de base du suivi d'instructions sémantiques : un dilemme entre l'exploration et l'exploitation.

Nous avons formulé ce problème comme un **processus décisionnel de Markov** basé sur des modèles stochastiques des instructions de l'opérateur et du détecteur d'objet. Nous avons modélisé les instructions en considérant les capacités de l'opérateur à évaluer mentalement des distances et des directions. En définissant la *distance de contournement*, nous avons modélisé ce qui peut être considéré comme une trajectoire rectiligne entre deux objets. Nous avons modélisé le détecteur d'objet en tenant compte des imperfections de la détection (objets non-détectés et détections parasites) et de la localisation. Un modèle du champ de vision, basé sur une carte d'occupation, permet également de tenir compte des obstructions. Finalement, nous avons défini l'*utilité des observations* pour prendre en compte la non-indépendance de résultats d'analyses de photos successives.

Nous avons présenté un **filtre à particules** pour une estimation rapide de la distribution de probabilité jointe sur les positions des objets, et donc de l'état du système. En partant de la méthode de *Sequential Importance Sampling*, nous avons présenté un mécanisme pour la génération à intervalles réguliers de nouvelles particules afin d'éviter la dégénérescence.

Nous avons proposé une **politique heuristique** pour la résolution du problème en exploitant sa dualité. D'une part, nous avons défini l'*utilité* de la recherche d'un objet donné en vue de

localiser le dernier objet. D'autre part, nous avons défini l'*utilité* d'une pose dans l'environnement pour localiser l'objet recherché. Cette politique heuristique nous a servi de politique de base à l'application de la méthode de **Monte Carlo Tree Search**, dont nous avons cherché à améliorer les performances dans le cadre spécifique de notre problème. Notamment, nous avons défini un contrôle de haut niveau basé sur la sélection d'un objectif et sa remise en question, nous avons utilisé un modèle simplifié pour la planification et nous avons proposé une méthode originale de *progressive widening* basée sur l'utilité heuristique des poses.

5.2 Limitations et pistes d'améliorations

Nous avons vu que la méthode de *Monte Carlo Tree Search* est d'autant plus efficace que le nombre d'explorations de l'arbre est important. Une des principales difficultés est donc d'implémenter des simulations de *rollout* suffisamment rapides. Dans notre cas, ceci s'est principalement traduit par une simplification du modèle du système. Si ce modèle simple permet d'obtenir des résultats pertinents, il repose néanmoins sur de grosses hypothèses et approximations : localisation parfaite, carte de l'environnement connue, unicité des objets.

Estimation et planification

Nous avons vu qu'il est possible d'utiliser un modèle simplifié pour la planification d'objectifs avec la méthode MCTS : notamment en réduisant le nombre de particules pour l'estimation d'état, en négligeant les erreurs de localisation des détections, ou encore en ne considérant aucune remise en question de l'objectif. Il pourrait être intéressant d'explorer davantage cette distinction entre un modèle précis pour l'estimation d'état et un modèle grossier pour la planification. Procéder de la sorte permettrait, d'une part, de modéliser précisément ce que le robot connaît à un instant donné de l'environnement sans compromettre, d'autre part, les performances de la méthode MCTS pour évaluer grossièrement les conséquences futures des actions applicables.

Environnement inconnu

Lorsque l'environnement est inconnu, comme dans l'exemple de l'étudiant perdu de l'introduction, l'intérêt du problème de suivi d'instructions sémantiques est évident. Maintenant, si une carte de l'environnement est fournie, nous perdons une partie significative des applications pratiques envisageables : pourquoi prendre du temps pour donner une série d'instructions au robot lorsqu'on pourrait simplement lui indiquer son objectif sur une carte ?

L'extension de notre solution au cas d'un environnement inconnu n'est pas triviale. La difficulté n'est pas dans la cartographie : l'application d'une méthode classique de *semantic localization and mapping* basée sur les données de la caméra ou d'un capteur laser permettrait de construire progressivement une carte d'occupation. Le problème est que, pour effectuer des simulations de *rollout*, il serait nécessaire de simuler cet environnement. Avec les positions des objets pour seules inconnues, simuler une transition d'état consiste simplement à échantillonner ces positions depuis la distribution correspondante, puis à simuler un résultat d'analyse étant donné ces positions. Avec un environnement inconnu chaque cellule de la carte d'occupation est une inconnue supplémentaire qu'il faudrait pouvoir échantillonner afin de simuler une transition d'état. Autrement dit, il faudrait être capable de générer des cartes d'occupation réalistes des parties inexplorées de l'environnement.

Modèle d'instruction

Notre modèle d'instruction évalue les capacité de l'opérateur du robot à évaluer mentalement des distances et des directions. Dans le cadre de ce travail, nous nous sommes contentés de décider empiriquement de valeurs conservatrices pour les paramètres de ce modèle. Une analyse statistique plus rigoureuse pourrait être mise en place. Par exemple, nous pourrions envisager de questionner un ensemble d'étudiants pour obtenir leurs évaluations de distances et de directions relatives à des éléments connus de leur établissement.

Discrétisation de l'environnement

Discrétiser l'environnement en le divisant à l'aide d'une grille entraîne une série de problèmes classiques. Au niveau de la modélisation du détecteur, un objet qui chevauche ou est à l'intersection entre plusieurs cellules pose problème pour la localisation ; et la modélisation des obstructions est d'autant moins précise que les cellules sont grandes. En définissant une carte de navigabilité grossière, il est possible de bloquer erronément certains passages étroits comme, par exemple, des portes.

Si cette discrétisation en cellules carrées présente l'avantage de la simplicité pour la rapidité des simulations, il pourrait être utile d'explorer des alternatives comme, par exemple, des cellules hexagonales pour la modélisation du détecteur d'objet, ou une triangulation de l'environnement pour la navigabilité.

Capacité d'observation

Dans notre problème simplifié, la capacité d'observation du robot est limité à l'identification d'objets de certaines classes (*chaise*, *porte*). Comme mentionné à l'introduction, il existe des méthodes permettant d'identifier d'autres types d'éléments dans l'environnement comme, notamment, des lieux (*couloir*, *cuisine*, *bureau*). Une autre approche intéressante à étudier serait l'identification de textes. Dans un environnement intérieur (*bureau*, *école*), ceci permettrait par exemple de s'aider de panneaux de directions ou de plaques identificatrices de locaux pour atteindre l'objectif.

5.3 Contraintes de mise en œuvre réelle

L'ensemble des résultats présentés dans ce travail ont été obtenus par simulation ; dans un premier temps à l'aide d'un modèle stochastique simplifié du robot et du détecteur d'objet, ensuite à l'aide de l'environnement de simulation *Gazebo*. La question qui se pose est alors celle de l'applicabilité de notre solution à un robot réel.

Carte et localisation parfaite

Comme mentionné à l'introduction, le robot peut se localiser dans la carte fournie de l'environnement à l'aide d'une méthode basée, par exemple, sur un filtrage des données issues de l'odométrie, d'un capteur laser et/ou de la caméra (*odométrie visuelle*). L'hypothèse de localisation parfaite est dans ce cas assez réaliste. Un élément qui n'a cependant pas été pris en compte est la présence d'obstacles supplémentaires par rapport à ce qui est indiqué sur la carte, notamment les objets recherchés. Sans changer fondamentalement notre méthode, il serait possible d'en prendre compte par une mise à jour dynamique des cartes d'occupation et de navigabilité.

Détecteur d'objets

Même si nos simulations sont assez réalistes grâce à l'utilisation de l'environnement de simulation *Gazebo*, la détection d'objet dans un environnement réel peut entrainer des difficultés supplémentaires. Tout d'abord, des photos réelles risquent de présenter d'avantages d'artefacts dus, notamment, à la luminosité et aux imperfections du capteur. Ensuite, un environnement réel contient généralement beaucoup d'objets et de caractéristiques particulières susceptibles de générer des détections parasites, contrairement aux environnements très *propres* de nos simulations.

5.4 Conclusion

Ce mémoire constitue une introduction à un problème original de suivi d'instructions sémantiques par un robot mobile. Présenter une formulation mathématique simplifiée de ce problème et en développer une solution nous ont permis d'en mettre en évidence des éléments clés. Des développements supplémentaires sont encore nécessaires pour étendre le domaine d'application de notre solution et en valider les résultats dans des situations réelles.

RÉFÉRENCES

N. Atanasov, B. Sankaran, J. Le Ny, G. Pappas, et K. Daniilidis, "Nonmyopic view planning for active object classification and pose estimation", *Robotics, IEEE Transactions on*, vol. 30, no. 5, pp. 1078–1090, Oct 2014.

N. Atanasov, M. Zhu, K. Daniilidis, et G. Pappas, "Semantic Localization via the Matrix Permanent", dans *Robotics : Science and Systems*, 2014.

P. Auer, N. Cesa-Bianchi, et P. Fischer, "Finite-time analysis of the multiarmed bandit problem", *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

H. Bay, T. Tuytelaars, et L. Van Gool, "SURF : Speeded Up Robust Features", dans *Computer Vision – ECCV 2006*. Springer Berlin Heidelberg, 2006, vol. 3951, pp. 404–417.

D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3e éd. Belmont, Massachusetts : Athena Scientific, 2005.

A. Borji et L. Itti, "State-of-the-art in visual attention modeling", *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 1, pp. 185–207, Jan 2013.

J. E. Bresenham, "Algorithm for computer control of a digital plotter", *IBM Syst. J.*, vol. 4, no. 1, pp. 25–30, Mars 1965.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton *et al.*, "A survey of Monte Carlo tree search methods", *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 1–43, 2012.

G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, et B. Bouzy, "Progressive strategies for Monte-Carlo tree search", dans *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*. World Scientific, 2007, pp. 655–661.

G. Chaslot, S. Bakkes, I. Szita, et P. Spronck, "Monte-Carlo tree search : A new framework for game ai." dans *AIIDE*, 2008.

J. D. Cohen, S. M. McClure, et A. J. Yu, "Should I stay or should I go ? how the human brain manages the trade-off between exploitation and exploration", *Philosophical Transactions of the Royal Society of London B : Biological Sciences*, vol. 362, no. 1481, pp. 933–942, 2007.

A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, et N. Bonnard, "Continuous upper confidence trees", dans *Learning and Intelligent Optimization*. Springer, 2011, pp. 433–445.

R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search", dans *Computers and games*. Springer, 2007, pp. 72–83.

N. Dalal et B. Triggs, "Histograms of oriented gradients for human detection", dans *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, June 2005, pp. 886–893 vol. 1.

E. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

A. Doucet et A. M. Johansen, "A tutorial on particle filtering and smoothing : Fifteen years later", *Handbook of Nonlinear Filtering*, vol. 12, pp. 656–704, 2009.

P. F. Felzenszwalb, R. B. Girshick, D. McAllester, et D. Ramanan, "Object detection with discriminatively trained part based models", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

S. Gelly et D. Silver, "Monte-Carlo tree search and rapid action value estimation in computer go", *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

L. Kocsis et C. Szepesvári, "Bandit based Monte-Carlo planning", dans *Machine Learning : ECML 2006*. Springer, 2006, pp. 282–293.

N. Koenig et A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator", dans *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.

T. L. Lai et H. Robbins, "Asymptotically efficient adaptive allocation rules", *Advances in applied mathematics*, vol. 6, no. 1, pp. 4–22, 1985.

D. Lowe, "Object recognition from local scale-invariant features", dans *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, 1999, pp. 1150–1157 vol.2.

D. Meger, P.-E. Forssén, K. Lai, S. Helmer, S. McCann, T. Southey, M. Baumann, J. J. Little, et D. G. Lowe, "Curious George : An attentive semantic robot", *Robotics and Autonomous Systems*, vol. 56, no. 6, pp. 503 – 511, 2008.

D. Nister et H. Stewenius, “Scalable recognition with a vocabulary tree”, dans *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2, 2006, pp. 2161–2168.

N. Otsu, “A threshold selection method from gray-level histograms”, *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

M. Pandey et S. Lazebnik, “Scene recognition and weakly supervised object localization with deformable part-based models”, dans *Computer Vision (ICCV), 2011 IEEE International Conference on*, Nov 2011, pp. 1307–1314.

A. Pronobis, B. Caputo, P. Jensfelt, et H. Christensen, “A discriminative approach to robust visual place recognition”, dans *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct 2006, pp. 3829–3836.

R. Rusu, N. Blodow, et M. Beetz, “Fast Point Feature Histograms (FPFH) for 3D registration”, dans *Robotics and Automation, 2009. ICRA ’09. IEEE International Conference on*, May 2009, pp. 3212–3217.

K. Shubina et J. K. Tsotsos, “Visual search for an object in a 3D environment using a mobile robot”, *Computer Vision and Image Understanding*, vol. 114, no. 5, pp. 535 – 547, 2010.

A. Torralba, K. Murphy, W. Freeman, et M. Rubin, “Context-based vision system for place and object recognition”, dans *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, Oct 2003, pp. 273–280 vol.1.

J. Uotila, M. Maula, T. Keil, et S. A. Zahra, “Exploration, exploitation, and financial performance : analysis of S&P 500 corporations”, *Strategic Management Journal*, vol. 30, no. 2, pp. 221–231, 2009.

S. Yitzhaki, “Gini’s mean difference : A superior measure of variability for non-normal distributions”, *Metron*, vol. 61, no. 2, pp. 285–316, 2003.