Processus Décisionnels de Markov pour l'autonomie ajustable et l'interaction hétérogène entre engins autonomes et pilotés

Mathieu Lelerre

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité INFORMATIQUE

Préparée au sein de l'Université de Caen Normandie

Processus Décisionnels de Markov pour l'autonomie ajustable et l'interaction hétérogène entre engins autonomes et pilotés

Présentée et soutenue par
Mathieu LELERRE

Thèse soutenue publiquement le 17/05/2018
devant le jury composé de

M. FRANÇOIS CHARPILLET	Directeur de recherche à l'INRIA, UNIVERSITE NANCY 1 HENRI POINCARE	Rapporteur du jury
M. FREDERIC VANDERHEAGEN	Professeur des universités, UNIVERSITE VALENCIENNES UVHC	Rapporteur du jury
M. LAURENT JEANPIERRE	Maître de conférences, UNIVERSITE CAEN NORMANDIE	Membre du jury
M. STEPHANE LECOEUCHE	Professeur des universités, ECOLE D'INGENIEURS DES MINES DE DOUAI	Président du jury
M. ABDEL ILLAH MOUADDIB	Professeur des universités, UNIVERSITE CAEN NORMANDIE	Directeur de thèse

Thèse dirigée par ABDEL ILLAH MOUADDIB, Groupe de recherche en informatique, image, automatique et instrumentation

UNIVERSITÉ CAEN NORMANDIE

MIIS

GREYC

Remerciements

Je tenais à rermercier Abdel-Illah Mouaddib pour l'investissement qu'il a apporté à ma thèse, que ce soit de la recherche d'investissement, de conférences ou de son soutien pour la rédaction. L'autonomie et la confiance qu'il m'a accordé ont été un grand confort, et malgré son emploi du temps chargé, il a été d'une grande aide pour cette thèse, que ce soit pour sa mise en place ou son déroulement. Merci pour tout.

Je remercie également Laurent Jeanpierre, qui a proposé de suivre la thèse en milieu de parcours. Son jugement et ses conseils ont permis une nette progression au cours de cette thèse.

Merci à la Direction Générale de l'Armement, Nexter Robotics et Dassault aviation pour leur participation financière ainsi que pour les échanges que j'ai pu avoir avec eux, qui ont pu m'apporter une vision différente de mes travaux de thèse.

Je tenais à remercier le laboratoire GREYC et l'Université de Caen Normandie pour m'avoir accueilli et formé depuis la licence jusqu'à la fin de ma thèse. Plus en particulier l'équipe MAD pour m'avoir inspiré, accueilli dans mon stage et ma thèse, et pour tous les conseils, le temps et les opportunités que ses membres m'ont accordé.

Enfin, je remercie Quentin, Fabio, Jonathan, Christopher, Loïs pour m'avoir supporté autant de temps dans le même bureau et pour tous les échanges qu'on a pu avoir. Je remercie aussi et les membres de l'Optic pour tous les moments de détente que nous avons pu avoir. Sans vous tous, cette thèse et ces années à Caen n'auraient pas été aussi plaisantes et enrichissantes.

Table des matières

Glossaire : Processus décisionnels de Markov et apprentissage

A	Liste d'actions dans un Processus Décisionnel de Markov (MDP).
S	Liste d'états dans un MDP.
T	Fonction de transition dans un MDP.
V'	Fonction de valeur obtenue après une mise à jour dans un algorithme, calculé à partir d'une fonction de valeur V.
V_π	Fonction de valeur de la politique π.
π^*	Politique optimale.
π	fonction attribuant pour chaque état une action à effectuer.
r	Fonction d'utilité dans un MDP.
s_t	Etat du système à un instant t.
s_{t+1}	(s_{t+1}) Etat du système à un instant t+1, c'est à dire après l'action planifiée depuis un état s_t.
IRL	Apprentissage par renforcement inverse.
MDP	Processus Décisionnel de Markov.
RL	Apprentissage par renforcement.

Glossaire : Estimation d'une politique

D_i^p	Domaine du paramètre i décrivant les informations fournies par l'opérateur sur les états du système.
D_i	Domaine de la variable i composant un état.
H_a	Historique des actions observées dans le cadre de l'observation d'un agent télé-opéré pour l'estimation de sa politique.
H_s	Historique des états atteints dans le cadre de l'observation d'un agent télé-opéré pour l'estimation de sa politique.
$Q_i(v)$	Préférence de l'opérateur sur la variable i lorsqu'elle prends la valeur v.
Q	Table d'apprentissage de `DIST` sur les états.
S_{est}	Liste des états atteignables à partir de l'action estimée d'un agent au cours d'une déviation.
S_{obs}	Liste des états atteignables à partir de l'action observée d'un agent au cours d'une déviation.
X	Ensemble de variables composant un état.
Δ	Distance entre deux états.
Q^+	Table de préférence de l'opérateur sur une variable d'état apprise par la méthode dist à partir de l'action observée lors d'une déviation.
Q^-	Table de préférence de l'opérateur sur une variable d'état apprise par la méthode dist à partir de l'action estimée lors d'une déviation.
δ	Distance maximale autorisée entre deux états pour effectuer une mise à jour avec la méthode `DIST`.
$\mathcal{D}_{ok}$	Fonction vérifiant que la distance entre deux états est inférieure à d_{max}.
$\mathcal{D}$	Fonction retournant la distance entre deux états.
π_{est}	Politique estimée.
π_{init}	Politique estimée initiale de l'opérateur.
a_o	Action observée depuis une déviation.
c	Coefficient d'apprentissage utilisé pour mettre à jour les politiques `LEARN`, `DIST`, ainsi que les méthodes dérivées.
k	Taille de l'historique pour l'apprentissage de `LEARN`, `DIST` et les méthodes qui en sont dérivées.
n_p	Nombre de paramètres définis par l'opérateur.
n	Nombre de variables composant un état.
$p_i(s)$	Valeur du paramètre i à l'état s.
r_d	Fonction de récompense générée suite àun apprentissage de type `DIST`.

r_l Fonction de récompense générée suite à un apprentissage de type `LEARN`.

s_o Etat atteignable dans le cadre d'une déviation, à partir de l'action observée choisie par l'opérateur.

t_h Indexage de l'historique des états et des actions dans le cadre de l'observation d'un agent télé-opéré pour l'estimation de sa politique. Il représente le nombre d'actions effectuées depuis une observation.

$x_i(s)$ Valeur de la variable i à l'état s.

`+CDIST` Méthode `DIST`, combinant un apprentissage des préférences sur les variables et les paramètres. Réduit les maximums locaux en n'appliquant que les malus appris sur les états (voir section 4.3.3).

`+CLEARN` Méthode `LEARN`, estimant la politique de l'opérateur en déduisant des préférences sur les variables composant un état, et sur les informations fournies par l'opérateur sur ces états. Réduit les maximums locaux en n'appliquant que les malus appris sur les états (voir section 4.3.3).

`+DISTP` Méthode `DIST`, apprenant des préférences sur les informations apportées par l'opérateur sur les états du système. Réduit les maximums locaux en n'appliquant que les malus appris sur les états (voir section 4.3.3).

`+DIST` Méthode `DIST`, estimant la politique de l'opérateur à partir de préférences déduites des actions observées. Réduit les maximums locaux en n'appliquant que les malus appris sur les états (voir section 4.3.3).

`+LEARNP` Méthode `LEARN`, estimant la politique de l'opérateur à partir des préférences sur les informations apportées par l'opérateur sur les états du système. Réduit les maximums locaux en n'appliquant que les malus appris sur les états (voir section 4.3.3).

`+LEARN` Méthode `LEARN`, estimant la politique à partir de préférences déduites des actions observées. Réduit les maximums locaux en n'appliquant que les malus appris sur les états (voir section 4.3.3).

`CDIST` Méthode `DIST`, combinant un apprentissage des préférences sur les variables et les paramètres.

`CLEARN` Méthode `LEARN`, estimant la politique de l'opérateur en déduisant des préférences sur les variables composant un état, et sur les informations fournies par l'opérateur sur ces états.

`DISTP` Méthode `DIST`, apprenant des préférences sur les informations apportées par l'opérateur sur les états du système.

`DIST` Méthode `DIST`, estimant la politique de l'opérateur à partir de préférences déduites des actions observées, avec réduction du sur-apprentissage.

`FORCE` Méthode `FORCE`, estimant la politique de l'opérateur à partir des actions observées.

`LEARNP` Méthode `LEARN`, estimant la politique de l'opérateur à partir des préférences sur les informations apportées par l'opérateur sur les états du système.

`LEARN` Méthode `LEARN`, estimant la politique à partir de préférences déduites des actions observées.

P Liste de paramètres décrivant les informations fournies par l'opérateur sur les états du système.

Glossaire : Semi-autonomie

A'	Liste des actions combinées entre contrôle du robot(A, A_a, A_o) .
A_a	Liste des actions d'interactions agent vers opérateur.
A_f	Liste des autorisations déjà rejetées..
A_o	Liste des actions d'interactions opérateur vers agent.
$A_{\mathfrak{R}}$	Liste des actions restreintes par la restriction $\mathfrak{R}$. Si cette liste est vide, toutes les actions sont concernées.
A_{ff}	Liste de liste des autorisations déjà rejetées dont il faut calculer une politique..
P_w	Fonction retournant la probabilité que l'opérateur accepte de surveiller l'agent à un état et pour une action donnés.
P_{OK}	Fonction retournant la probabilité que l'opérateur accepte que l'agent effectue une combinaison état-action restreinte par une restriction de type `REQUEST_REQUIRED`.
$P_{\texttt{TO}}$	Fonction retournant la probabilité que l'opérateur accepte de télé-opérer l'agent à un état donnée.
S'	Espace d'état étendu du restricted MDP.
S_{OK}	États atteignables en cas d'accord lorsque l'agent effectue une requête de type `REQUEST_REQUIRED`.
$S_{\mathfrak{R}}$	Liste des états restreints par la restriction $\mathfrak{R}$. Si cette liste est vide, tous les états sont concernés.
$S_{\neg OK}$	États atteignables en cas de refus lorsque l'agent effectue une requête de type `REQUEST_REQUIRED`.
T_{OK}	Fonction de transition associée aux accords d'une requête de type `REQUEST_REQUIRED`.
$T_{\neg OK}$	Fonction de transition associée aux refus d'une requête de type `REQUEST_REQUIRED`.
T_{auth}	Fonction de transition associée aux demandes de permissions.
T_a	Fonction de transition associée aux changements de niveau d'autonomie requis l_r.
T_o	Fonction de transition associée aux changements de niveau d'autonomie observé l_o.
T_r	Fonction de transition associée aux contrôles du robot.
V_{A_f}	Fonction de valeur associée à une liste A_f d'autorisation déjà refusées.
$\mathcal{R}$	Liste des restrictions du système.
$\mathfrak{R}$	Une restriction.
π_o	Politique estimée des interactions de l'opérateur avec l'agent.
π_{A_f}	Politique calculée en considérant la liste A_f des autorisations déjà refusées.
$\pi_{\mathfrak{R}}$	Politique par défaut de $\mathfrak{R}$ à exécuter si l'agent tente une combinaison état-action restreinte par $\mathfrak{R}$, sans respecter les conditions de la restriction.
$\pi_{\texttt{TO}}$	Fonction retournant la probabilité que l'opérateur, durant la télé-opération, choisisse une action donnée à un état donné.
a_{def}	Action par défaut remplaçant l'action que l'aget essaierait de faire lorsque celle-ci viole ses restrictions.
l_o	Niveau d'attention de l'opérateur observé.
l_r	Niveau d'attention de l'opérateur requis par l'agent.

req_i	Action pour demander à l'opérateur l'accord de faire une action a dans un état s de la restriction numéro i de type `REQUEST_REQUIRED`.
$RL_{\mathfrak{R}}$	Niveau de la restriction $\mathfrak{R}$.
`Att`	Niveau d'attention de l'opérateur.
`RL`	Niveau d'une restriction.
`TO`	Action d'attente de télé-opération.
auth	Numéro de restriction accordée.

Introduction

Contexte

Les robots vont être de plus en plus utilisés dans les domaines civils comme militaires. Le domaine civil contient la robotique de service, domestique, logistique et médical. Dans le domaine militaire. on trouve la sécurité, la surveillance et l'intervention. Ces robots, opérant en flottes, peuvent accompagner des soldats au combat, ou accomplir une mission en étant supervisés par un poste de contrôle. Du fait des exigences d'une opération militaire, il est difficile de laisser les robots décider de leurs actions sans accord d'un opérateur ou surveillance, en fonction de la situation. Nous présentons ci-dessous plusieurs exemples de scénarios présentant ce type de contraintes.

— Dans le cadre de la surveillance maritime, des véhicules aériens sans pilotes (UAV) sont envoyés patrouiller le long des côtes afin de détecter des navires suspects. Lorsqu'un suspect est repéré, l'UAV ne peut évidemment pas intervenir directement sans autorisation au préalable. Les opérateurs sont donc en mesure de prendre le contrôle de l'UAV afin d'intervenir.
— Dans le cas d'une opération d'infiltration militaire, des robots terrestres peuvent-être envoyés pour le reconnaissance, la surveillance ou l'intervention. Lorsque l'un d'entre eux repère quelque chose, celui-ci doit en informer son opérateur, qui peut être un des soldats qu'il accompagne. S'il ne parvient pas à identifier ce qu'il a repéré, il peut par exemple demander à l'opérateur d'observer le flux vidéo des caméras afin que l'opérateur lui dise quoi faire.
  De même, l'opérateur peut observer une zone suspecte et télé-opérer un robot pour observer cette zone.
— Au cours d'une mission de secours, il est possible d'envoyer des robots terrestres pour rechercher des survivants, ou apporter du matériel aux secouristes. Néanmoins, les terrains pouvant être accidentés, il est possible que des zones soient trop dangereuses pour laisser le robot faire le trajet seul. Dans ce cas, un opérateur prend la main pour le déplacer jusqu'à une zone sécurisée. De même, les conditions peuvent empêcher le robot de reconnaître efficacement une personne à secourir. Dans ce cas, il faudra demander à un opérateur de surveiller les caméras du robot afin de s'assurer que personne n'est oublié.

Ces exemples démontrent l'intérêt pour un robot de pouvoir changer son niveau d'autonomie, c'est à dire sa dépendance vis à vis de son opérateur pour décider de ses actions. On parle alors d'autonomie ajustable, ou adaptable, et c'est cette notion qui est au cœur de cette thèse.

Thèse

Dans cette thèse, nous cherchons plusieurs moyens d'appliquer l'autonomie ajustable dans des applications telles que décrites ci-dessus, afin de déployer des robots assez autonomes pour assister des personnes au cours d'une mission via des méthodes de planification, mais également de s'assurer que le robot respecte des règles morales, ou ne compromette la mission, par exemple en tirant sur des civils. En permettant ainsi au robot d'être piloté, les opérateurs sont assurés de pouvoir reprendre la main en cas d'imprévu, et d'assurer qu'aucun risque d'ordre matériel, humain, éthique ou sur le bon déroulement de la mission ne soit prise de manière imprévisible, ou en cas de difficultés prévisibles au cours de la mission.

Dans cette thèse, nous nous attardons sur deux problématiques :

1. D'une part, nous cherchons à exploiter l'autonomie ajustable de sorte à ce qu'un robot puisse accomplir sa mission de la manière la plus efficace possible, tout en respectant des restrictions assignées par un opérateur sur son niveau d'autonomie. Pour cela, celui-ci est en mesure de définir pour un ensemble d'états et d'actions donné un niveau de restriction. Ce niveau peut par exemple imposer au robot de ne pas utiliser de lui même une arme embarquée, mais de demander à un opérateur de prendre le contrôle du robot afin de d'utiliser cette arme, ou de demander la surveillance par un opérateur avant de s'aventurer dans une zone à risque. Nous présenterons dans cette thèse des approches visant à intégrer ce type de restrictions au cours de la planification des actions du robot.
2. D'autre part, comme nous envisageons la possibilité que plusieurs robots soient déployés en même temps, ces robots doivent se coordonner pour accomplir leurs objectifs. Lorsque tous les agents sont autonomes, il suffit de planifier les actions de ces agents en même temps, ou en fonction de la planification des autres agents pour assurer la coordination. Seulement, comme les opérateurs peuvent prendre le contrôle de certains d'entre eux, la question de la coordination se pose. En effet, l'opérateur ayant ses propres préférences, perceptions de l'environnement, connaissances et étant sujet aux hésitations, dues par exemple au stress, il est difficile de prévoir les actions que celui-ci va effectuer, et donc de s'y coordonner.

   Nous proposerons dans cette thèse une approche visant à estimer la politique exécutée par un robot télé-opéré à partir d'apprentissage basé sur les actions observés de ce robot.

La notion de planification est très présente dans ces travaux. Ceux-ci se baseront sur des modèles de planifications comme les Processus Décisionnels de Markov, qui sont des modèles représentant les états, les actions et l'incertitude de l'influence de ces actions sur l'environnement d'un agent.

Au cours de nos travaux, nous présenterons des extensions des Processus Décisionnels de Markov pour permettre d'estimer la politique d'un robot télé-opéré, puis pour calculer une politique respectant les restrictions imposées par l'opérateur.

Projet GARDES

Au cours d'une mission militaire, aujourd'hui, des robots peuvent être déployés afin d'aider les soldats. Ceux-ci permettent aux soldats de prendre le contrôle du robot pour la reconnaissance, ou d'effectuer une mission de manière autonome en parallèle avec eux. Néanmoins, ceux-ci sont principalement télé-opérés, afin de s'assurer que le comportement du robot ne nuise pas à la mission. En effet, les interventions militaires comportent des risques pour les soldats, il faut donc une certaine assurance sur le comportement d'un robot pour pouvoir se fier à lui.

FIGURE 1 – Photo d'un robot nerva(source : `http://www.army-guide.com/eng/product4978.html`)

Le projet ANR/DGA GARDES est un projet en collaboration entre le GREYC, la DGA et NEXTER-ROBOTICS. Celui-ci a pour but de permettre aux soldats de profiter de la présence des robots sans pour autant nécessiter une surveillance ou une prise de contrôle permanente. Pour cela, le laboratoire travaille avec des robots NERVA, un robot développé par NEXTER-ROBOTICS, afin d'y intégrer des solutions d'autonomie ajustable. Un des robots est présenté en figure 1.

Ces robots, pouvant fonctionner de manière autonomes, ou pilotés, permettent d'appliquer différentes solutions d'autonomie ajustable. Contrairement à cette thèse, pour assurer le contrôle des agents autonomes, ceux-ci reçoivent des recommandations, telles que des états désirables, d'autres à éviter. [Mouaddib *et al.*, 2015] proposent un modèle permettant de représenter ces recommandations, visant à orienter les actions du robot afin qu'il reste, dans la mesure du possible, dans les états souhaités.

Ce projet fait face à la même problématique sur la coordination entre agents avec différents niveaux d'autonomies, est également présente dans ce projet. De plus, bien que l'opérateur n'est pas supposé intervenir de la même façon dans le contrôle de l'agent, la proposition de cette thèse visant à définir des restrictions sur l'autonomie de l'agent n'est pas incompatible avec celle proposant d'influencer le comportement de l'agent. Les résultats seront donc utilisés dans le cadre de ce projet. Cela permettra, ainsi, de mettre en place les modèles présentés dans le cadre de cette thèse sur des robots, et dans des problèmes réels.

Organisation du document

Ce document est composé de 3 parties.

Première partie : Etat de l'art

La première partie, présentant l'état de l'art, est composée de 3 chapitres.

— Le premier chapitre présente ce qu'est l'intelligence artificielle, un agent, un système multi-agent et un problème de planification.

— Le second introduit les Processus Décisionnels de Markov, des algorithmes connus pour les

résoudre, l'apprentissage, ainsi qu'une extension pour prendre en compte l'observabilité et la planification multi-agents.
— Le troisième chapitre définit la semi-autonomie et présente quelques méthodes permettant d'évaluer un système semi-autonome.

Seconde Partie : Contributions

La seconde partie présente les contributions de cette thèse. Elle est composée de deux chapitres.

— Le quatrième chapitre concerne l'estimation de la politique d'un agent télé-opéré. Dans ce chapitre, nous présenterons trois méthodes à base d'apprentissage :
  — `FORCE`, une méthode d'apprentissage d'une politique observée, ayant pour principe de retenir la dernière action effectuée dans chaque état, et à adapter la politique pour les états non atteints en conséquence.
  — `LEARN` est une méthode de prédiction apprenant les préférences de l'opérateur sur les états.
  — `DIST` est une version d'apprentissage locale de `LEARN` visant à limiter le sur-apprentissage.

  Nous avons également présenté dans ce chapitre des améliorations de ces méthodes, comme la suppression d'objectifs parasites ou l'intégration d'informations renseignées par l'opérateur.
— Le cinquième chapitre présente les modèles proposés pour intégrer des restrictions à la politique d'un agent à autonomie ajustable.

Troisième partie : Expérimentations

La troisième partie contient les expérimentations visant à valider les différentes méthodes proposés.

— Le chapitre 6 décrit les conditions d'expériences et les résultats obtenus sur l'estimation de politique, ainsi que les observations obtenues à partir de l'implémentation sur les robots nervas.
— Le chapitre 7 concerne l'intégration de restrictions.

On finit la thèse par une conclusion et des perspectives.

Première partie

État de l'art

Introduction de la Partie 1

Un système autonome est doté d'une capacité de raisonner, planifier puis décider pour agir d'une manière indépendante d'un opérateur. Pour cela, la formalisation d'un système autonome se fonde sur plusieurs concepts théoriques que nous allons introduire dans cette partie. En effet, nous allons nous focaliser sur le domaine de la planification mono et multi-agent sous différentes hypothèses, comme l'observabilité, l'incertitude et la communication et les différents modes d'autonomie afin d'éviter les limites d'une approche d'autonomie ou de télé-opération totale.

L'objectif de cette partie est d'introduire les concepts et les outils utilisés au cours de la thèse. Dans le premier chapitre, nous introduisons les concepts d'intelligence artificielle et introduirons le problème de planification dans les cadres mono et multi-agent.

Dans le second chapitre, nous présentons des modèles de planification, comme les Processus Décisionnels de Markov ainsi que les algorithmes de résolution les plus connus de l'état de l'art. Nous introduirons brièvement l'observation partielle, puis nous attarderons sur la planification multi-agents, à la planification à initiative -mixte et à l'apprentissage. Enfin, nous parlerons des limites des agents autonomes.

Dans le troisième chapitre, nous parlerons d'autonomie. Nous expliquerons l'intérêt des agents semi-autonomes, et les modèles proposés ainsi que leurs limites. Nous présenterons également différents niveaux d'autonomie ajustable, d'un point de vue planification et d'un point de vue exécution. Nous parlerons également d'autonomie ajustable et d'évaluation de systèmes semi-autonomes.

Chapitre 1

Définitions : Agents et environnement pour la planification

Sommaire

L'objectif de cette thèse est de permettre à un ensemble de robots d'assister un robot téléopéré, et de demander l'assistance d'un opérateur. Pour cela, nous présentons dans ce chapitre la problématique de la planification. Dans un premier temps, nous introduirons l'intelligence artificielle, puis nous parlerons de la modélisation d'un agent dans son environnement, et enfin nous évoquerons les systèmes multi-agents, ainsi que les problématiques associées.

1.1 Intelligence artificielle

L'intelligence artificielle (IA) est ce qui permet à une entité, ou agent, de résoudre des problèmes résolus par les hommes à l'aide d'un raisonnement. Les IA sont de plus en plus présentes dans la vie de tous les jours. Nous la retrouvons dans différents domaines, comme :

— **les jeux vidéos :** Elle y est utilisée pour animer les personnages non joueurs. Elle permet à ces personnages auparavant animés par des scripts d'avoir une meilleure adaptation face à l'adversaire humain. Deux exemples notables sont la victoire d'IA face aux champions du monde d'échec et de GO.

— **Robotique :** L'IA en robotique est utilisée pour animer des robots de manière autonome. Elle permet aux robots, à partir d'un objectif donné par un opérateur de planifier ses actions en fonction de leur perception de leur environnement par leurs capteurs (caméra,

micro, ...). Par exemple, un robot peut servir de guide dans un magasin en escortant, à leur demande, les clients aux rayons qu'ils souhaitent atteindre. A partir de la connaissance du bâtiment et de l'objectif défini par le client, le robot calcule une stratégie visant à l'emmener là ou le client le souhaite. D'autres exemples de robotique de surveillance et de sécurité existent aussi [Ishikawa et Suzuki, 1997] [Paruchuri *et al.*, 2008] [Vorobeychik *et al.*, 2012].

— **Planification logistique :** L'IA est utilisée pour gérer la mise en place d'un système de production, avec gestion de ressources. [Picard *et al.*, 2017] étudient par exemple un système permettant le déploiement de taxis autonomes dans une ville afin d'optimiser les trajets de ces véhicules en fonction des demandes des utilisateurs.

— **Aide à la décision :** L'IA contribue également à l'aide à la décision dans de nombreux domaines, comme la médecine, la finance, ... Par exemple, certains magasins s'appuient sur des systèmes d'aides à la décision, couplés avec la base de données des achats des clients, pour inférer des couples de produits achetés ensembles, afin de gérer la disposition des produits dans les rayons.

L'intelligence artificielle est un domaine vaste, et répond à plusieurs problématiques, comme le traitement automatique de langues, l'apprentissage ou le raisonnement. Dans notre cas, nous nous intéressons à celui de la planification, permettant à une IA de déterminer à l'avance les actions à effectuer en fonction de la situation.

1.2 Problème de planification

La planification est une question en intelligence artificielle qui s'est posée dès l'aube de ce domaine. Elle consiste en la modélisation du problème, afin que l'agent puisse établir une stratégie en vue de résoudre son objectif.

1.2.1 Paramètres du modèle

Pour modéliser un problème de planification, il faut à la fois modéliser les agents qui le composent, ainsi que les éléments extérieurs à ces agents parmi lesquels ils évoluent. Ces éléments extérieurs constituent l'environnement de l'agent.

Les agents

Le système est d'abord constitué d'agents. Plusieurs définitions d'un agent existent en fonction des besoins du système.

Dans notre cas, nous considérons une définition proposée par [Russell et Norvig, 2003] :

Définition 1. *Un agent est simplement une entité qui agit.*

Selon cette définition, un agent peut être une personne, un robot, ou encore un programme. Cet agent a une certaine **autonomie**.

Définition 2. *L'autonomie d'un agent est sa capacité de choisir ses actions indépendamment d'une autre entité. Ainsi, un agent autonome est indépendant, à l'opposé d'un agent télé-opéré, où une autre entité choisit ses actions.*

L'agent, et son environnement, ont un **état**.

Définition 3. *Un état est la situation de l'agent et de son environnement à un instant donné.*

En général, l'état est constitué de l'ensemble des éléments pertinents pour la décision de l'agent.

L'agent peut avoir des connaissances sur cette situation. Ces connaissances sont déduites des **observations** de l'agent, issues des capteurs, qui peuvent informer complètement ou partiellement sur la situation.

Les agents dans l'environnement peuvent agir. Ils peuvent modifier l'état du système. Il faut donc modéliser les **actions**, et les conséquences de celles-ci pour déterminer la stratégie de l'agent.

L'agent a un objectif à accomplir. Pour cela, il peut agir de différentes façons, mais toutes n'auront pas la même efficacité en fonction du but recherché par celui-ci. Par exemple, lorsqu'une personne prend sa voiture, elle peut choisir le chemin le plus court, ou le moins consommateur d'énergie. Ces critères doivent être pris en compte dans le modèle.

Propriétés de L'environnement

Un environnement peut avoir certaines propriétés, dont les suivantes, décrites par [Russell et Norvig, 2003] :

Observable ou non : En fonction de la perception de l'agent sur l'environnement, celui-ci peut être en mesure de le percevoir totalement (par exemple une partie du jeu d'échec). On qualifie ce type d'environnement de *totalement observable*. Dans ce cas, l'agent perçoit directement l'état courant du système. Dans le cas contraire, l'environnement est dit *partiellement observable*. Par exemple, un robot qui n'est équipé que d'une caméra frontale ne verra que devant lui.

Déterministe ou Stochastique : Les actions des agents ont une conséquence sur leur état ainsi que sur l'environnement. Lorsque, à partir d'un état, les conséquences d'une action sont connues à l'avance (par exemple un coup aux échecs), l'environnement est dit déterministe. On parle d'environnement stochastique lorsqu'on connaît la distribution de probabilité des états atteints, ou la probabilité pour chaque état que l'environnement se retrouve dans celui-ci après une action d'un agent (par exemple, jouer à pile ou face). Il est également possible de connaître les états atteignables à partir d'une action, sans pour autant connaître la probabilité d'atteindre chacun d'entre eux.

Statique ou dynamique : Un environnement est dynamique s'il peut évoluer indépendamment des actions des agents au cours de l'exécution ou de la planification (par exemple une porte qui peut s'ouvrir ou se fermer à cause d'un courant d'air). Dans le cas contraire, où l'environnement ne peut changer spontanément, celui-ci est statique.

Discret ou continu : L'environnement est dit continu s'il y a un nombre infini de situations possibles dans cet environnement. Par exemple, un robot dont la position est définie à partir de coordonnées x,$y \in \mathbb{R}$ a une infinité de situations possibles dans cet environnement. Quand le nombre d'état est fini, l'environnement est alors discret. Par exemple, un robot dont la position est définie par la pièce dans laquelle il se trouve dans un bâtiment a un nombre fini de situations possibles.

Ouvert ou clos : Un environnement est ouvert à partir du moment où des événements imprévisibles, et donc non modélisables, peuvent perturber les agents ou l'environnement.

Mono ou multi-agent : L'environnement est qualifié de mono-agent ou de multi-agent en fonction du nombre d'agents évoluant dans celui-ci.

Episodique ou séquentiel : L'environnement est dit épisodique lorsque le choix des actions des agents ne dépend que de l'instant courant. Dans un environnement séquentiel, les décisions passées influent sur la décision courante.

Ces différentes propriétés influent sur la complexité du problème. Il arrive que la modélisation soit simplifiée pour faciliter la planification. Il est par exemple possible de discrétiser un environnement continu. Par exemple, au lieu d'identifier la position d'un agent sur des coordonnées réelles, il est possible d'identifier la pièce dans laquelle celui-ci se trouve, ou de décomposer son espace en cases, et d'identifier sa position sur la case correspondante. Néanmoins, il y a alors perte d'informations sur l'état du système, ce qui peut nuire à l'efficacité de la stratégie calculée, mais en contre-partie cette simplification du modèle facilite la résolution du problème.

1.2.2 Exemple : Navigation dans un bâtiment sécurisé

Pour illustrer les différentes propriétés de l'environnement, considérons l'exemple d'un bâtiment à risque nucléaire. Du point de vue de la sécurité, l'entrée de personnes est limitée. Un robot est déployé pour récupérer les relevés de radio-activité en temps réels grâce à ses capteurs. Son opérateur définit, au cours de la mission, des points à observer. Le robot doit alors se rendre à ceux-ci pour faire ses mesures.

Pour cela, le robot doit se déplacer. Il est en mesure de pivoter vers la gauche ou la droite de 90°, d'avancer, de reculer ou de rester sur place.

Nous supposons notre robot équipé de capteurs lui permettant de se localiser de manière précise dans l'environnement. La seule mission du robot étant de se déplacer afin de récupérer ses mesures, les valeurs de ces dernières n'ont pas d'impact sur la stratégie de l'agent. Il n'est donc pas nécessaire de les modéliser dans l'environnement. Nous considérons alors le système composé du robot, de sa position géographique et des points à observer, totalement observables. De plus, nous considérons le robot suffisamment précis pour que chacune de ses actions ne le mène qu'à un état donné. Nous considérons l'environnement statique, clos et mono-agent puisque seul celui-ci y intervient. Il est également séquentiel, puisque l'agent doit suivre une séquence d'actions pour atteindre son objectif. L'environnement est également infini, puisque la position du robot est continu. Néanmoins, comme expliqué dans la section précédente, il est possible de discrétiser la position, l'environnement serait alors fini dans le modèle.

1.3 Introduction aux systèmes multi-agents

Dans un système multi-agent(SMA), plusieurs agents agissent en même temps dans l'environnement, et peuvent interagir entre eux. [Weiss, 1999] présente de nombreux domaines associés aux systèmes multi-agents, comme la formation de coalition, la coordination par vote, par négociation... Nous nous intéressons plus dans cette thèse à la planification. Quand nous parlons de planification pour des systèmes multi-agents, il est nécessaire de se poser quelques questions, comme celles détaillées ci-dessous.

1.3.1 Connaissance et observabilité des agents

En fonction du système à modéliser, les agents ne sont pas forcément en mesure d'observer les autres agents dans l'environnement. Dans l'exemple décrit en section 1.2.2, nous pourrions

considérer que plusieurs robots sont envoyés pour prendre les relevés, partiellement observables entre eux. On peut aussi considérer qu'à chaque instant, chaque agent envoie son état aux autres pour simuler l'observation totale, mais avec l'hypothèse de la disponibilité permanente de la communication, et l'hypothèse que l'ensemble des observations des agents constitue une observation totale de l'état du système.

1.3.2 Types d'agents

Les agents déployés dans l'environnement n'ont pas forcément le même objectif, ce qui peut influencer la nature des interactions entre les agents.

Les agents coopératifs vont avoir un objectif commun, et vont tenter de maximiser l'utilité de l'ensemble des agents. Par exemple, dans une mission de reconnaissance, les agents peuvent se répartir l'exploration du terrain pour en couvrir le maximum en économisant le coût en énergie. [Panagou et Kumar, 2014] proposent un exemple d'application sur une flotte de robots évoluant de manière coopérative en formation dans un environnement avec obstacles.

Les agents "égoïstes" vont chercher à maximiser leur récompense, sans prendre en compte les objectifs des autres. Par exemple, un agent pourrait bloquer l'accès d'un autre agent à l'un de ses objectifs pour accomplir plus facilement le sien.

Les agents en confrontation sont des agents qui cherchent à la fois à maximiser leur gain, mais également à minimiser le gain des autres. Par exemple, [Vorobeychik *et al.*, 2012] décrivent un scénario où un agent cherche à s'introduire dans une zone surveillée par un autre.

1.3.3 Types d'interactions

Dans un SMA, les agents peuvent influer sur les autres, et donc perturber la politique d'autres agents, par exemple en coupant la route de l'un d'entre eux. Plusieurs types d'interactions existent pour éviter ce genre de désagréments :

- La coordination : Il est possible de planifier des tâches coordonnées afin d'assurer la cohérence de l'ensemble du système. [Lesser, 1999] présente une introduction aux problèmes de coordination dans les systèmes multi-agents.
- La communication : Les agents peuvent communiquer leurs intentions ou leur plan pour que les autres agents puissent s'y adapter. [O'Brien et Nicol, 1998] présente la Foundation for Intelligent Physical Agents [1], une organisation ayant proposé une standardisation d'agents communicant entre eux. Plusieurs travaux se sont basés sur ces standards, comme le framework JADE, proposé par [Bellifemine *et al.*, 1999].
- La négociation : Au cours de la communication , il peut arriver que deux agents se nuisent en suivant leur plan. Il est alors possible pour eux de négocier leurs objectifs (ou les moyens de les atteindre) pour mettre à jour leur politique et éviter les perturbations. [Kraus, 1997]

1.3.4 Ouvert ou fermé

On dit qu'un système multi-agent est ouvert lorsque de nouveaux agents peuvent y entrer et d'autres peuvent le quitter durant l'exécution. Dans un problème fermé, le modèle du système n'a pas à prendre en compte l'entrée ou sortie d'agents du système, et les agents n'ont pas besoin de s'y adapter en changeant leur politique.

Dans un système ouvert, l'arrivée d'un agent peut avoir une forte incidence sur les autres agents. Il faut soit que la politique calculée reste valide malgré l'ajout d'un agent, soit être en

1. `http://www.fipa.org`

mesure de la recalculer. De même, cela peut influencer l'espace d'états de l'agent, qui peut se retrouver augmenté par la localisation de l'agent dans l'environnement si celui-ci est totalement ou partiellement observable. Dans le cas de retrait d'un agent, les politiques peuvent changer pour prendre en charge sa mission dans un cadre coopératif. On peut également le considérer partiellement observable, afin de ne pas avoir à retenir toutes les informations de tous les agents actuels et à venir dans l'espace d'état.

1.3.5 Hétérogène ou homogène

Un SMA est dit homogène si tous les agents du système ont les même fonctionnalités. Dans le cas contraire, on dit qu'il est hétérogène.

Dans le cadre de cette thèse, on introduit également à l'autonomie hétérogène. C'est à dire que les différents agents n'ont pas, tous, le même niveau d'autonomie, ou la capacité à agir indépendamment d'un opérateur, ou pilote.

1.4 Conclusion

Dans ce chapitre, nous avons défini ce qu'est une IA, et présenté des domaines d'applications de l'IA, comme dans le cadre logistique ou de la planification robotique.

Nous avons également défini les propriétés d'un problème de planification, comme les agents, les états, l'environnement et ses propriétés (comme son observabilité, s'il est déterministe ou stochastique, ...).

Nous avons également parlé de systèmes multi-agents, et de différentes propriétés associées aux systèmes multi-agents, comme l'observabilité, les types d'interactions ou l'objectivité des agents. Dans cette thèse, nous étudions des agents coopératifs dans un environnement observable. En effet, chacun des agents peut librement communiquer son état avec les autres.

Dans le chapitre suivant, nous parlerons de méthodes de planifications. Nous y introduirons notamment les Processus Décisionnels de Markov, ainsi que certaines extensions permettant de modéliser un environnement partiellement observable ou un système multi-agent.

Chapitre 2

Planification, décision et apprentissage

Sommaire

Nous avons vu dans le chapitre précédent ce qu'était un agent et présenté divers éléments d'un problème de planification, comme les agents et leur environnement.

La planification est un domaine important dans la recherche en intelligence artificielle. C'est avec celle-ci que les robots sont animés en vu d'accomplir un objectif. La planification consiste à modéliser le problème auquel l'agent est soumis afin de calculer une stratégie optimale permettant de résoudre ce problème.

Dans ce chapitre nous présenterons en premier lieu une première méthode de modélisation pour planifier l'exemple défini en 1.2.2. Ensuite, nous expliquerons en quoi la modélisation de

l'incertitude est nécessaire. Nous parlerons alors des Processus Décisionnels de Markov, permettant la planification sous incertitude.

Comme les agents ne sont pas toujours dans des situations leurs permettant de se situer exactement dans leur environnement, nous présenterons ensuite les Processus Décisionnels de Markov Partiellement Observables, permettant à un agent de calculer une stratégie en fonction de ses actions précédentes et des observations qu'il a faites sur son environnement.

Par la suite nous parlerons des difficultés pour une personne de modéliser de manière exacte l'environnement du robot, ou des contraintes associées à la planification. Nous présenterons alors l'apprentissage, qui permet à un robot d'apprendre une stratégie à l'aide d'un opérateur humain, ou au cours de la mission. Nous détaillerons l'apprentissage par renforcement et l'apprentissage par renforcement inverse.

Une fois ces sujets traités, il reste à parler de la planification avec plusieurs agents. Nous aborderons la planification centralisée, qui consiste à calculer en même temps les politiques de tous les agents, et la planification décentralisée, consistant à calculer les politiques des agents séparément, en les faisant s'adapter aux autres agents.

2.1 Modélisation de l'exemple de navigation dans un bâtiment sécurisé

Plusieurs solutions existent pour modéliser un problème de planification. Plusieurs d'entre elles, comme la formulation STRIPS [Fikes et Nilsson, 1971], se basent sur un modèle d'état et d'actions pour représenter les connaissances de l'agent dans son environnement. Nous présentons comment modéliser simplement le système avec un espace d'état, un espace d'action, une fonction de transition et un critère de performance.

2.1.1 L'espace d'états

L'agent doit être en mesure de se situer dans son environnement, et de voir l'évolution de ce dernier pour planifier ses actions. Il dépend donc de son état, tel que défini en définition 3. Nous considérons alors l'espace d'état contenant l'ensemble des états possibles du système.

Dans notre exemple de navigation présenté en 1.2.2, le robot se déplace dans son environnement. L'état est constitué de la position de l'agent ainsi que son orientation. Nous la modéliserons à partir de coordonnées cartésiennes et de son orientation, définie par les points cardinaux. Plus formellement, dans notre exemple, notre état est un tuple $\langle x,y,o\rangle$, où (x,y) est la position du robot, et o son orientation. Notre espace d'état contient donc l'ensemble des combinaisons de positions et d'orientations possibles du robot.

2.1.2 Actions

Pour modéliser l'influence de l'agent sur son environnement, et son évolution jusqu'à son objectif, il est nécessaire d'identifier les actions de l'agent ainsi que leur impact. Nous pouvons considérer qu'agir dans l'environnement implique un changement d'état. Par conséquent, nous définissons la **fonction de transition** :

Définition 4. *Une fonction de transition $T : S \times A \to S$ retourne l'état atteint s_{t+1} en effectuant l'action a_t à l'état s_t*

Cette fonction permettra à l'agent de connaître l'impact de ses actions sur l'environnement.

s_t	$T(s_t, \uparrow)$	$T(s_t, \curvearrowleft)$	$T(s_t, \curvearrowright)$	$T(s_t,\emptyset)$
$\langle x,y,N\rangle$	$\langle x,y+1,N\rangle$	$\langle x,y,W\rangle$	$\langle x,y,E\rangle$	$\langle x,y,N\rangle$
$\langle x,y,S\rangle$	$\langle x,y-1,S\rangle$	$\langle x,y,E\rangle$	$\langle x,y,W\rangle$	$\langle x,y,S\rangle$
$\langle x,y,W\rangle$	$\langle x-1,y,W\rangle$	$\langle x,y,S\rangle$	$\langle x,y,N\rangle$	$\langle x,y,W\rangle$
$\langle x,y,E\rangle$	$\langle x+1,y,E\rangle$	$\langle x,y,N\rangle$	$\langle x,y,S\rangle$	$\langle x,y,E\rangle$

Tableau 2.1 – Table de transition pour l'exemple cité en 1.2.2, pour toute position (x,y) de l'agent dans l'espace

Dans notre modèle, une action changera la position ou l'orientation de l'agent dans son environnement. Ces actions seront énumérées de la façon suivante :

— $\uparrow$: Avancer tout droit.
— $\curvearrowleft$: Rotation dans le sens anti-horaire.
— $\curvearrowright$: Rotation dans le sens horaire.
— $\emptyset$: rester sur place.

La table 2.1 présente la table de transition pour notre exemple.

Pour choisir l'action à effectuer, il faudra que l'agent évalue les conséquences de celle-ci.

2.1.3 Critère d'évaluation

Plusieurs solutions existent pour évaluer l'impact des actions de l'agent, comme par exemple les chances de réussite d'une mission. Ici, nous introduirons la notion d'utilité, correspondant aux préférences vis à vis des états qu'il atteint, ou des préférences sur les actions possible pour chaque état de l'espace d'états.

Définition 5. *La fonction d'utilité correspond à la combinaison de la satisfaction de l'agent et du coût de l'action de l'agent.*

La fonction d'utilité $U(s,a)$ peut donc être exprimée à partir d'une fonction de satisfaction (ou de récompense) $r : S \times A \to \mathbb{R}$, ainsi que d'une fonction de coût $c : S \times A \to \mathbb{R}$ de la façon suivante :

$$U(s,a) = r(s,a) - c(s,a) \tag{2.1}$$

Dans notre exemple, l'agent peut avoir une récompense en atteignant un objectif, mais avoir un coût en se déplaçant.

Ainsi, notre agent est en mesure de prédire l'influence de ses actions dans un état. Il est capable d'assigner un score à une action, et donc de choisir une action par rapport aux autres.

2.1.4 Résolution

Le choix des actions de l'agent va se faire selon son critère d'évaluation de celles-ci. Il cherchera à maximiser ce critère pour déterminer la meilleure action à accomplir.

De manière naïve, l'agent peut déterminer les actions une par une en choisissant celle qui aura la meilleure récompense sur l'état actuel. Dans ce cas, selon notre exemple, si l'agent est éloigné de l'objectif, il ne pourra pas l'atteindre en une seule action, et comme bouger a un coût, il restera sur place.

Pour éviter ce phénomène, l'agent doit planifier plusieurs coups à l'avance. Pour cela, l'agent doit simuler à l'avance le résultat de ses actions pour les évaluer.

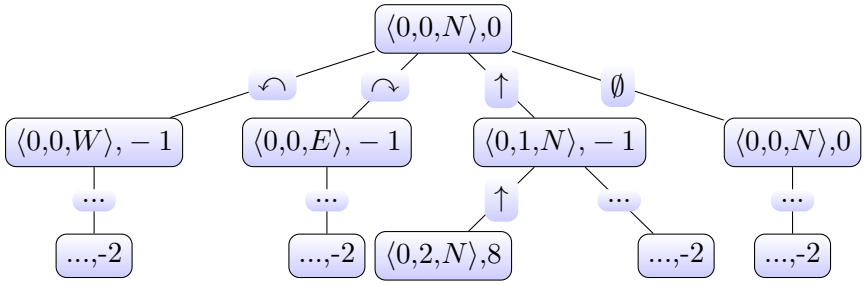


FIGURE 2.1 – Exemple de parcours d'arbre pour une séquence de deux actions.

Une méthode consiste à construire un arbre. Chaque arête de l'arbre correspond à une action effectuée. Chaque noeud a deux valeurs. La première est l'état atteint en exécutant la séquence d'actions obtenue à partir des arêtes parentes. La seconde est la valeur d'utilité du chemin parcouru pour atteindre le noeud courant. Celle-ci est obtenue en additionnant toutes les utilités des transitions parentes. L'algorithme 1 présente la méthode pour le construire.

Algorithme 1 : Construction d'un arbre de planification

Données : S, A, T, U, s_0

1 Ar arbre avec pour racine $\leftarrow \langle s_0,0\rangle$;
2 f file initialisée à $\{\langle s_0,0\rangle\}$;
3 **tant que** $\langle$*Condition de fin*$\rangle$ **faire**
4 $\langle s_t,u\rangle \leftarrow f.defiler()$;
5 **pour** $a \in A$ **faire**
6 $s_{t+1} \leftarrow T(s_t,a)$;
7 $u' = u + U(s_t,a)$;
8 $Ar.ajouterfils(\langle s_t,u\rangle,a,\langle s_{t+1},u'\rangle)$;
9 $f.enfiler(\langle s_{t+1},u'\rangle)$;

Les deux premières lignes servent à initialiser la construction de l'arbre avec la racine. Ar est l'arbre, et f est une file permettant le parcours en largeur de l'arbre. La ligne 6 permet de récupérer les états suivants s_{t+1} pour chaque action disponible pour l'agent, à partir d'un état donné s_t. L'utilité u' du parcours permettant d'atteindre s_{t+1} est calculée en ligne 7, en sommant l'utilité de la transition actuelle avec l'utilité du parcours menant à s_t. Enfin, le couple $\langle s_{t+1}\rangle$ est ajouté à l'arbre, connecté à s_t grâce à une arrête d'action a. Ce noeud est ensuite ajouté à la file pour analyser les actions suivantes. Le parcours est arrêté lorsque la condition en ligne 3 est remplie, comme par exemple un nombre d'itérations maximum, ou jusqu'à ce que chaque feuille de l'arbre corresponde à l'objectif.

La figure 2.1 représente la génération d'un arbre, reprenant l'exemple décrit en 1.2.2. Nous considérons que l'agent démarre à l'état $\langle 0,0,N\rangle$. Pour limiter le développement de l'arbre, l'objectif se situe directement en $\langle 0,2\rangle$.

Pour conclure sur l'optimalité de la planification, il faudrait parcourir toutes les branches de l'arbre jusqu'à ce que toutes atteignent l'objectif. Or, il existe des séquences d'actions ne permettant pas de l'atteindre. Par exemple, un robot restant sur place ou tournant autour de lui-même. Pour éviter de planifier à l'infini, nous considérons la notion d'**horizon**, qui est le nombre maximum d'actions considérées à l'avance au cours de la panification. On dit alors qu'une politique est optimal pour un horizon h.

Une fois l'arbre construit, pour retrouver la stratégie optimale, il suffit de regarder les feuilles de l'arbre et de sélectionner celle qui a la plus haute valeur d'utilité. En remontant de la feuille avec la plus grande utilité vers la racine de l'arbre, on obtient les actions à accomplir pour chaque état.

Cette méthode permet ainsi de calculer une politique pour un agent évoluant dans un environnement déterministe. Seulement, dans de nombreux cas, les conséquences d'une action ne sont pas toutes déterministes (par exemple le lancé de dé), et ce modèle ne permet alors plus de planifier dans ces condition. C'est pourquoi, dans la section suivante, nous introduisons les Processus Décisionnels de Markov, permettant de modéliser des environnements stochastiques.

2.2 Processus Décisionnels de Markov

Les processus décisionnels de markov (MDP) sont des modèles mathématiques utilisés pour résoudre des problèmes de décision. Ils permettent de modéliser l'environnement stochastique d'un agent, ainsi que les actions à disposition de celui-ci, afin de planifier les actions à effectuer en fonction de la situation courante à chaque instant.

2.2.1 Description du modèle

Un MDP est défini comme un tuple $\langle S,A,T,r\rangle$, où
- S est l'espace d'état du système.
- A est l'espace d'actions de l'agent.
- $T : S \times A \times S \to [0;1]$ est la fonction de transition. Elle retourne la probabilité d'atteindre l'état $s' \in S$ en effectuant une action donnée $a \in A$ dans un état donné $s \in S$.
- $r : S \times A \to \mathbb{R}$ représente la fonction d'utilité définie en 2.1.3.
- Dans certains MDP, il est aussi possible de trouver un état de départ s_0, correspondant à l'état d'origine de l'agent avant toute action.

La fonction de transition diffère de celle de la définition 4. En effet, pour un environnement stochastique, plusieurs états sont atteignables pour une action et un état donnés. Néanmoins, une distribution de probabilités doit respecter certaines contraintes :

$$\forall s_t \in S,\forall a \in A, \sum_{s_{t+1}\in S} T(s_t,a,s_{t+1}) = 1 \tag{2.2}$$

Il est notable que cette notation est également valide dans un environnement déterministe. Cette fonction aura alors la propriété suivante :

$$\forall s_t \in S,\forall a \in S,\exists s_{t+1} \in S,T(s_t,a,s_{t+1}) = 1,\forall s \neq s_{t+1} T(s_t,a,s) = 0 \tag{2.3}$$

Un MDP contient donc tous les éléments nécessaires pour résoudre un problème déterministe ou stochastique.

Au cours de l'exécution, l'agent sait être dans un état s_t. Il y effectue une action a_t, et en déduit le nouvel état s_{t+1} de l'environnement via ses capteurs.

A partir d'un MDP, il est possible de calculer une *politique* π, définissant la stratégie de l'agent.

Définition 6. *Une politique est une fonction qui assigne à chaque état une action à effectuer. Elle peut être à horizon fini, et dépendre du nombre d'actions disponibles ($\pi : S,N \to A$), ou infini, et ne dépendre que de l'état courant ($\pi : S \to A$).*

s_t	a	s_{t+1}	$T(s_t,a,s_{t+1})$
$\langle x,y,N,* \rangle$	$\curvearrowleft$	$\langle x,y,W,* \rangle$	1
$\langle x,y,N,* \rangle$	$\curvearrowright$	$\langle x,y,E,* \rangle$	1
$\langle x,y,N,* \rangle$	$\emptyset$	$\langle x,y,N,* \rangle$	1
$\langle x,y,N,false \rangle$	$\uparrow$	$\langle x,y+1,N,* \rangle$	1
$\langle x,y,N,true \rangle$	$\uparrow$	$\langle x,y+1,N,* \rangle$	0.9
		$\langle x-1,y+1,N,* \rangle$	0.05
		$\langle x+1,y+1,N,* \rangle$	0.05

Tableau 2.2 – Table de transition de l'exemple en section 1.2.2, représentant la probabilité de transition depuis un état s_t vers un état s_{t+1} en effectuant une action a.

2.2.2 Exemple de modèle de navigation de robot dans un bâtiment sécurisé avec environnement stochastique

Reprenons l'exemple proposé en 1.2.2. Considérons maintenant que le sol est glissant à certains endroits. Dans ces conditions, le robot, en avançant, risque de se déplacer sur le côté en même temps.

Pour intégrer cet élément dans notre modèle, nous ajoutons l'information dans l'espace d'état. Ainsi, un état est noté $\langle x,y,r,g \rangle$, où $g \in \mathbb{B}$ est vrai si le sol est glissant en position (x,y). Seulement, en considérant que le robot n'a pas d'influence sur l'état du sol, comme le sol ne peut être que glissant ou non à une position donnnée, le nombre d'état reste le même. Autrement dit :

$$\forall r \in \{N,S,O,E\} \nexists x,y \in \mathbb{R} \text{ tel que } \langle x,y,r,true \rangle \in S \text{ et } \langle x,y,r,false \rangle \in S. \tag{2.4}$$

L'espace d'actions ne change pas dans cet exemple. En revanche, la fonction de transition n'étant plus déterministe, il est nécessaire de la redéfinir.

Considérons qu'en temps normal, le robot atteint la position désirée comme dans le cas déterministe, et que sur une zone glissante, il a 90% de chances d'atteindre la position désirée, et 10% de se retrouver à gauche ou à droite. La fonction de transition obtenue est décrite en table 2.2, dans le cas où le robot est orienté vers le nord avant la transition. Seules les transitions possibles y sont représentés. Il est facile de généraliser les autres directions en changeant la position et l'orientation de l'état suivant. Le symbole '*', pour la variable indiquant l'état de glissement, indique que celle-ci peut prendre n'importe quelle valeur.

2.2.3 Résolution

Résoudre un MDP revient à calculer une politique pour l'agent. D'après [Bellman, 1956], toute politique optimale est composée de sous-politiques optimales. Ainsi, calculer une politique optimale revient à calculer les actions optimales pour chaque état. Pour cela, nous calculons l'utilité de ces actions pour chaque état.

Soit $V_\pi : S \to \mathbb{R}$ la fonction de valeur de la politique, correspondant à la récompense espérée de la politique π. Elle est calculée à partir de l'équation 2.5 proposée par [Bellman, 1956].

$$\forall s_t \in S, V_\pi(s_t) = r(s_t,\pi(s_t)) + \gamma \sum_{s_{t+1} \in S} T(s_t,\pi(s_t),s_{t+1}) \times V_\pi(s_{t+1}) \tag{2.5}$$

La valeur espérée représente donc la récompense obtenue en effectuant une action dans un état, ajoutée à l'espérance des gains futurs. γ est un facteur d'atténuation permettant de régler

l'attention que porte l'agent aux actions futures. Proche de 0, les états et actions futures sont presque ignorés. Ce facteur permet ainsi de négliger les actions lointaines, et donc de calculer à horizon infini.

Définition 7. *Une politique est dite optimale, et notée π^*, si elle maximise la récompense espérée. Autrement dit, $\forall s \in S, \nexists \pi, V_\pi(s) > V_{\pi^*}(s)$.*

Autrement dit, pour calculer la politique optimale, il faut trouver la séquence d'actions maximisant la récompense espérée. Ceci peut être fait en résolvant l'équation suivante adaptée de l'équation de Bellman :

$$\forall s_t \in S, \pi^*(s_t) = \underset{a \in A}{\operatorname{argmax}} \left(r(s_t,a) + \gamma \sum_{s_{t+1} \in S} T(s_t,a s_{t+1}) V^{\pi^*}(s_{t+1}) \right) \tag{2.6}$$

Afin de résoudre cette équation, nous nous intéressons aux deux algorithmes fréquemment utilisés pour résoudre des MDP : *Value iteration* et *Policy iteration*.

Value Iteration(VI)

Value Iteration est un algorithme proposé par [Bellman, 1956]. C'est le plus utilisé pour calculer la fonction de valeur optimale d'un MDP, ainsi que la politique correspondante. Elle est décrite par l'algorithme 2.

Algorithme 2 : Value Iteration

Données : MDP $\langle S,A,T,r \rangle$, facteur d'atténuation γ

`/* Initialisation */`

pour $s \in S$ **faire**

  $V(s) = 0$;

`/* Planification */`

tant que $\langle Condition \rangle$ **faire**

  pour $s_t \in S$ **faire**

    $V'(s_t) \leftarrow \max_{a \in A} \left(r(s_t,a) + \gamma \sum_{s_{t+1} \in S} T(s_t,a,s_{t+1}) V(s_{t+1}) \right)$;

    $\pi(s_t) \leftarrow \operatorname{argmax}_{a \in A} \left(r(s_t,a) + \gamma \sum_{s_{t+1} \in S} T(s_t,a,s_{t+1}) V(s_{t+1}) \right)$;

  $V \leftarrow V'$;

retourner V,π;

Cet algorithme calcule, en ligne 5 et 6, la fonction d'utilité de chaque état, ainsi que la politique associée, à partir d'une politique précédemment calculée. Ainsi, l'idée de cet algorithme est de calculer la politique maximisant l'espérance de gain estimée à partir des précédentes estimations, jusqu'à satisfaire la condition d'arrêt de l'algorithme.

Plusieurs solutions existent pour cette condition. Nous en donnons deux exemples :

ϵ - approximation : La mise à jour de la politique s'arrête à partir du moment où la différence d'espérance $\Delta = \max_{s \in S} |V'(s) - V(s)|$ calculée entre deux itérations est en dessous d'un seuil $\Delta_{min} = \frac{\epsilon(1-\gamma)}{2\gamma}$. Selon [Bertsekas *et al.*, 1995], cette méthode assure la convergence de la politique vers une fonction de valeur ϵ-optimale. On dit qu'une politique calculée ainsi est à horizon infini.

Horizon fini : Une autre solution consiste à effectuer H itérations de la politique. H est alors appelé l'horizon, et correspond au nombre d'actions planifiées à l'avance pour décider de chaque état. La politique générée ainsi est dite optimale à un horizon H. Chaque itération de l'algorithme permet d'augmenter l'horizon de planification. L'avantage de cette méthode est un calcul plus rapide de la politique. Dans le cas d'une politique à horizon fini, la politique π_t peut être différente pour chaque pas de temps t.

De manière générale, la complexité de l'algorithme dépendra du nombre d'itérations à effectuer. La complexité d'une itération, elle, est de $O(|S|^2.|A|)$. [Sutton et Barto, 1998a] a expliqué que les ordinateurs en 1998 étaient en mesure de résoudre des MDP à 1 million d'états.

Policy Iteration(PI)

Contrairement à VI, qui calcule d'abord la fonction de valeur optimale, puis la politique associée, PI calcule une politique, puis l'améliore à partir de la fonction de valeur associée. Cet algorithme, proposé par [Weiss, 1960], est décrit par l'algorithme 3.

Algorithme 3 : Policy Iteration

Données : MDP $\langle S,A,T,r\rangle$, facteur d'atténuation γ

`/* initialisation */`

1 **pour** $s_t \in S$ **faire**
2 $\pi(s_t)$ choisi arbitrairement;
3 **répéter**
4 $\pi' \leftarrow \pi$;
5 Calculer V';
6 **pour** $s_t \in S$ **faire**
7 **pour** $a \in A$ **faire**
8 **si** $r(s_t,a) + \gamma \sum_{s_{t+1} \in S} T(s_t,a,s_{t+1}) \times V(s_{t+1}) > V(s_t)$ **alors**
9 $\pi(s_t) = a$;
10 **jusqu'à** $\pi = \pi'$;
11 **retourner** π ;

Dans un premier temps, une politique est initialisée de façon arbitraire. Il peut s'agir d'une politique aléatoire, comme d'une politique basée sur l'action la plus rentable à horizon 1 (sans prévoir les coups suivants). Par la suite, l'algorithme entame une série d'évaluations, puis d'améliorations de la politique jusqu'à ce que la politique ne puisse plus être améliorée. L'évaluation se fait via l'équation de Bellman (2.5), dans la boucle à la ligne 5. Le calcul de la politique s'arrête lorsque celle-ci ne peut plus être améliorée. L'amélioration de la politique se fait par la boucle en ligne 6, en cherchant une action qui améliore la fonction de récompense calculée plus tôt.

Si la politique a changé, alors elle est réévaluée, puis améliorée à nouveau. Si, au cours d'une itération, la politique n'a pas changé, alors elle ne changera plus par la suite, puisque la fonction de valeur ne changera pas. La politique obtenue est alors optimale.

Cet algorithme permet de calculer la politique en moins d'itérations. Néanmoins, les itérations sont plus longues. Il est difficile d'en juger une préférable parmi VI et PI. [Littman, 1996] explique que le choix parmi les deux algorithmes dépend de la structure du MDP.

2.3 Observation Partielle

Bien que nous ne traitons pas dans cette thèse de l'observation partielle, elle est une part importante du domaine des MDP. En effet, ceux-ci considèrent un environnement totalement observable. Or, dans une grande majorité des cas, l'observabilité est partielle. Par exemple, un robot de surveillance doit se déplacer pour observer une pièce. C'est pourquoi nous introduisons le concept.

Les Processus Décisionnels de Markov Partiellement Observables (POMDP),sont une extension des MDP, en vue d'ajouter la capacité d'observation de l'agent dans l'environnement.

Un POMDP est un tuple $\langle S,A,T,r,\Omega,O\rangle$, où :

— S est l'espace d'état,
— A est l'ensemble des actions disponibles,
— $T : S \times A \times S \to [0..1]$ est la fonction de transition,
— $r : S \times A \to \mathbb{R}$ est la fonction d'utilité,
— Ω est l'ensemble des observations possibles de l'agent,
— $O : S \times A \times S \times \Omega \to [0..1]$ est la fonction d'observation de l'agent. Celle ci permet, pour un agent transitant d'un état s_t à un état s_{t+1} en effectuant l'action a, de connaître la probabilité d'obtenir une observation o_t.

Contrairement à un MDP, à l'exécution, l'agent ne récupère pas comme information de ses capteurs un état s_t, mais une observation o_t. Il faut donc que l'agent prenne des décisions à partir de l'ensemble de ses actions et de ses observations, afin d'estimer l'état du système. Cela revient donc pour l'agent à calculer sa politique à partir d'un historique d'actions et d'observations. Hors, pour la planification, cela revient à représenter tous les historiques possibles, ce qui est complexe au vu de la taille possible d'un historique.

[Bertsekas *et al.*, 1995] a proposé de raisonner sur un état de croyance b_t, c'est à dire une distribution de probabilité sur les états à un instant t. Ainsi, $b_t(s)$ est la probabilité pour le système d'être dans l'état s à un instant t. Il démontre que cet état de croyance suffit pour prendre en compte l'ensemble de l'historique des observations. L'agent calcule alors sa politique à partir de l'état de croyance, de ses actions et de l'observation. Au cours de l'exécution, l'agent a une croyance b_t sur son état et effectue une action a. De par ses capteurs, il reçoit une observation selon la probabilité définie en équation 2.7.

$$\omega(b,a,o) = Pr(o|,b,a) = \sum_{s_t \in S} \sum_{s_{t+1} \in S} O(o|s_t,a,s_{t+1}) T(s_t,a,s_{t+1}) b(s_t) \tag{2.7}$$

Etant donné un état de croyance b, une action a, et une observation o, il est possible de déterminer l'état de croyance b_o^a, défini par l'équation 2.8. Les détails de développement de l'équation sont expliqués dans [Sigaud et Buffet, 2013].

$$\begin{aligned} b_o^a(s_{t+1}) &= Pr(s_{t+1}|b,a,o) \\ &= \frac{O(o|s_{t+1}) \sum_{s_t \in S} T(s_t,a,s_{t+1}) b(s_t)}{\sum_{s_t \in S} \sum_{s' \in S} O(o|s_t,a,s') T(s_t,a,s') b(s_t)} \end{aligned} \tag{2.8}$$

A partir de cette fonction, il est possible d'obtenir une fonction de transition sur les états de

croyances à l'aide de l'équation 2.9.

$$\begin{aligned} &\Phi(b,a,b') = Pr(b'|b,a) = \sum_{o \in \Omega} \omega(b,a,o)\delta(b',b_o^a) \\ &Avec\ \delta(x,y) = \begin{cases} 0 & si\ x = y \\ 1 & sinon \end{cases} \end{aligned} \tag{2.9}$$

Ne se basant plus sur les espace d'états mais sur les espaces de croyance, il faut définir la fonction de récompense $\rho(s,a)$ associée à un état de croyance et une action. Celle -ci est représentée par l'équation 2.10, et correspond à la somme des récompenses sur les états sur la probabilité d'être dans cet état.

$$\rho(b_t,a) = \sum_{s \in S} r(s,a)b_t(s) \tag{2.10}$$

La fonction de valeur de la politique n'est pas calculée de la même façon, du fait que l'agent ne connait pas l'état précis du système. L'agent calculera pour chaque état la fonction de valeur de celui-ci pour chaque action possible, et choisira l'action à effectuer en fonction de sa croyance sur ces états. Plus formellement, on définit pour chaque action a un alpha vecteur $\alpha_a = \{V_{a_1}, \ldots, V_{a_k}\}$ contenant pour chaque état la fonction de valeur associée en effectuant cette action.

Ainsi, la politique optimale se calcule à partir de l'équation 2.11.

$$V^*(b) = \max_{a \in A} \rho(b,a) + \sum_{o \in O} \Phi(b,a,b_o^a)V^*(b_o^a) \tag{2.11}$$

D'après [Papadimitriou et Tsitsiklis, 1987], à horizon fini, tandis que la complexité de la résolution d'un MDP est P-complet, celle d'un POMDP est PSPACE-complet. A horizon infini, d'après [Madani *et al.*, 1999], celle-ci est EXP(Ω,A,H,S), donc indécidable.

Il existe plusieurs variantes de POMDP. Par exemple, en général, un POMDP a un état de croyance initial b_0, mais un processus décisionnels de Markov partiellement observables possibilistes qualitatifs(π-POMDP), présenté par [Drougard *et al.*, 2015], est un POMDP sans connaissance initiale du problème. Il propose également un autre modèle, un processus décisionnel de Markov à observabilité mixte, permettant de simplifier le problème du π-POMDP en considérant certaines variables des états complètement observables.

2.4 Apprentissage

La planification permet de calculer la politique d'un agent avant la mission. Seulement, il peut arriver qu'un manque d'informations sur le système, comme la fonction de récompense, ou la fonction de transition complique la procédure de planification. Une méthode pour contourner ce problème est l'apprentissage.

L'apprentissage permet à un agent de décider de ses actions au cours de l'exécution de la politique, sans avoir à planifier ses actions à l'avance. On parle alors de planification en ligne. L'intérêt de cette approche est d'une part, de pouvoir déployer des agents dans des environnements inconnus, afin qu'ils apprennent comment y agir de manière efficace. D'autre part, cette approche permet d'éviter certaines difficultés lors de la modélisation du système. Par exemple, la configuration de la fonction de récompense peut être fastidieuse pour générer le comportement souhaité, et il peut être difficile de représenter toutes les transitions possibles dans la fonction de transition, ou d'évaluer les probabilités de ces transitions.

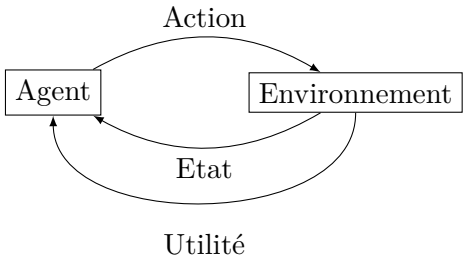


FIGURE 2.2 – Représentation de l'apprentissage par renforcement

Le domaine de l'apprentissage dans la recherche est vaste. Plusieurs finalités sont attendues de l'apprentissage, comme l'acquisition de connaissance. Par exemple, il est possible d'y retrouver l'extraction de connaissance, visant à déduire des connaissances d'un texte ([Mooney et Bunescu, 2005])ou d'un document web ([Alani *et al.*, 2003]). On peut également y retrouver l'agrégation de connaissances, visant à générer de nouvelles connaissances en suivant une logique associée. Par exemple, [Cohen-Solal *et al.*, 2015] présente une algèbre pour raisonner dans le temps, et [Rao et Georgeff, 1991] présente une architecture permettant à un agent de raisonner sur les connaissances et croyances d'un ensemble d'agents rationnels.

Dans notre cas, nous nous intéressons à l'apprentissage pour la décision. Nous y distinguons plusieurs familles :

- L'**apprentissage supervisé**, où l'opérateur peut influencer la stratégie de l'agent en interagissant avec lui, comme par exemple avec des signaux ([Knox et Stone, 2008]).
- L'**apprentissage par imitation** consiste à montrer à l'agent le comportement souhaité, afin qu'il puisse l'apprendre et le répéter. [Schaal *et al.*, 2003] présente ce type d'apprentissage comme solution pour reproduire des mouvement complexes.
- L'**apprentissage par renforcement**, présenté par [Sutton et Barto, 1998b], permet à un agent d'optimiser sa stratégie en comparant itérativement le résultat de ses actions.
- L'**apprentissage par renforcement inverse**, initié par [Ng et Russell, 2000], permet à un agent qui en observe un autre de calculer une fonction de récompense associé à la politique observée. Il est alors possible d'utiliser cette fonction de récompense pour calculer une nouvelle stratégie dans un espace d'état différent.

Nous détaillons ci-dessous l'apprentissage par renforcement, car celle-ci est souvent la base d'un apprentissage visant un comportement optimal, et l'apprentissage par renforcement inverse, car celui-ci se rapproche d'un apprentissage de la politique d'un opérateur, ce qui permettrait ainsi de se coordonner avec lui.

2.4.1 Apprentissage par renforcement

Principe

L'apprentissage par renforcement, présenté par [Sutton et Barto, 1998b] est une méthode permettant à un agent de s'adapter à son environnement sans intervention humaine. La figure 2.2 représente le fonctionnement classique de l'apprentissage par renforcement. Le temps du système est discrétisé en instant, qui sont organisés de la façon suivante :

L'exécution est décomposée, comme dans le cadre des MDP, en instants, où à chaque instant l'agent est à un état. Un instant se déroule de la manière suivante :

- L'agent effectue une action suivant sa stratégie, modifiant ainsi l'état de son environnement.

— L'agent, après chaque action, reçoit le nouvel état de l'environnement et de l'utilité de son action.
— Il apprend de ces informations une nouvelle stratégie.

Dans le cadre de l'apprentissage supervisé, l'agent reçoit une récompense de l'opérateur. IL est aisé ainsi pour l'opérateur de guider l'agent vers le comportement qu'il souhaite, par exemple avec une récompense positive pour toute action désirée, et négative dans le cas contraire.

Dans le cas de l'apprentissage par renforcement, l'agent reçoit sa récompense de l'environnement. Ainsi, l'agent n'est pas directement guidé vers la politique souhaité, et l'agent doit multiplier les essais et erreurs afin de trouver l'action qui maximisera l'utilité moyenne de la mission. Pour cela, l'agent apprend de l'utilité reçue une table associant état et actions à une utilité moyenne. Il suffit alors à l'agent de choisir son action de sorte à maximiser la moyenne obtenue. Il s'agit de la phase **exploitation** de la méthode.

Calcul d'une politique

La phase d'exploitation vise à maximiser l'utilité espérée en effectuant l'action. Pour combler le manque de connaissances de bases du modèle, et pour calculer l'utilité moyenne d'une action au cours de l'exécution, celle-ci apprend de l'utilité perçue à partir de l'environnement. Plusieurs méthodes d'apprentissage s'appliquent sur des MDP, comme Temporal Difference, ou TD, TD-learning, State–action–reward–state–action ou SARSA ([Sutton et Barto, 1998b]), Q-Learning ([Watkins et Dayan, 1992b]), Actor-Critic ([Barto *et al.*, 1983]), Dyna ([Sutton, 1991]), Queue-Dyna ([Peng et Williams, 1993]) ou encore la méthode sweeping ([Moore et Atkeson, 1993]).

Nous présentons SARSA, régulièrement utilisé en apprentissage, ci-dessous.

State–action–reward–state–action SARSA est un algorithme d'apprentissage présenté par [Sutton et Barto, 1998b], visant à évaluer les actions effectuées à chaque état. Elle attribue à chaque état et chaque action un score, noté $Q(,sa)$, où Q est la table d'apprentissage, remplie en fonction des récompenses reçues par l'agent à chaque couple d'état et action.

Après chaque action a_t effectuée à un état s_t, l'agent récupère de l'environnement une utilité u_t et un état atteint s_{t+1}, et met à jour sa table d'apprentissage en suivant l'équation 2.12, composée des variables suivantes :

— α, le coefficient d'apprentissage, influe sur la capacité d'apprentissage de l'algorithme. A 0, l'algorithme n'apprend pas et garde la table d'apprentissage initiale. A 1, l'agent enregistre la dernière valeur d'utilité reçue et ignore les précédentes.
— γ est un coefficient d'atténuation, influençant la prise en compte des actions futures. A 0, l'agent ne prend pas en compte les actions suivantes, tandis qu'à 1, l'action suivante a autant d'influence sur l'apprentissage que l'action courante.

$$Q'(s_t,a_t) = Q(s_t,a_t) + \alpha(u_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)) \tag{2.12}$$

Exploitation et Exploration

La phase exploitation permet à l'agent d'assurer une utilité moyenne de ses actions. Néanmoins, celle-ci n'est pas forcément optimale. En effet, en ne faisant que de l'exploitation, à la première action estimée meilleure que les autres à un état, l'agent n'en changera pas pour cet état tant que son utilité n'aura pas descendue, étant donné qu'il ne testera plus les autres actions. Pour éviter ce problème, l'apprentissage par renforcement contient également une phase

d'**exploration**, consistant à choisir des actions différentes de l'optimale pour tester de nouvelles solutions.

Néanmoins, il n'est pas possible de tout le temps explorer, sous peine de diminuer l'utilité des actions de l'agent. Ce dernier doit alors faire le compromis entre l'exploration et l'exploitation. Pour établir ce compromis, plusieurs solutions existent, indépendantes de la méthode d'exploitation choisie. Elles sont classées en deux catégories selon [Thrun, 1992].

1. Les méthodes indirectes, comme celles présentés par [Bertsekas et Tsitsiklis, 1995], utilisent peu d'informations issues de l'apprentissage. On y trouve par exemple les méthodes ci-dessous :
   — Exploiter pendant n_1 itérations lors de l'exécution, puis n_2 exploitations
   — $\epsilon - greedy$: cette méthode consiste à choisir entre l'exploration et l'exploitation aléatoirement. Ainsi, si le nombre tiré aléatoirement est inférieur à une variable ϵ, alors l'agent explore, sinon il exploite.
2. Les méthodes directes incluent dans la table $Q(s,a)$ un bonus associé à l'exploration. On y trouve par exemple les deux méthodes ci-dessous :
   — La méthode *recency-based* accorde un bonus défini par $\epsilon\sqrt{\delta n_{s_a}}$, où $\epsilon \in [0,1[$ est un coefficient augmentant l'exploration quand sa valeur est élevée. δn_{s_a} est le nombre d'itération depuis la dernière exécution de l'action a à l'état s. Ainsi, elle vise à explorer les actions non utilisées depuis longtemps, pour une utilité équivalente.
   — La méthode *uncertainty estimation* accorde un bonus de valeur $\frac{c}{n_{sa}}$, où c est le facteur influençant l'exploration, et n_{sa} est le nombre de fois où l'action a a été choisie à l'état s. Cette méthode vise donc, pour une utilité équivalente, à explorer les actions les moins souvent choisies.

Extensions d'apprentissage par renforcement

Bien que l'apprentissage par renforcement permette à un agent de choisir sa stratégie indépendamment d'un humain, un opérateur peut souhaiter influencer le comportement obtenu par cet apprentissage, pour l'accélérer par exemple, ou pour changer le comportement de l'agent.

[Knox et Stone, 2010] propose par exemple de combiner TAMER [Knox et Stone, 2008], un apprentissage supervisé, et de l'apprentissage par renforcement, comme SARSA. Il bénéficie ainsi des avantages de l'apprentissage supervisé en début d'apprentissage, où l'opérateur informe l'agent sur sa satisfaction de ses actions. L'agent tend ainsi plus rapidement vers une politique proche de l'optimale et a moins besoin d'explorer qu'avec un apprentissage par renforcement. Il bénéficie aussi des avantages de l'apprentissage par renforcement en fin d'apprentissage, avec une meilleure précision sur l'utilité espérée de chaque action.

[Abel *et al.*, 2017] propose quant à lui un framework permettant à un opérateur d'intégrer le processus de décision de l'agent, en lui permettant de recevoir et d'émettre des recommandations à l'agent. Il peut par exemple modifier la fonction d'utilité du modèle pour influencer les actions de l'agent, entraîner l'agent dans un simulateur, ou supprimer des actions, pour par exemple simplifier l'exploration. Il peut également choisir les actions que l'agent doit effectuer.

[Riley et Veloso, 2004] propose de répondre à un problème différent. Il considère le cas où aucune information sur le système n'est disponible. L'agent doit alors construire l'espace d'état et les fonctions de transition à partir des observations dont il dispose.

[Enjalbert et Vanderhaegen, 2017] combine l'apprentissage par renforcement et l'apprentissage profond ([Schmidhuber, 2015]) pour estimer l'efficacité d'un système où Hommes et machines coopèrent. L'apprentissage par renforcement permet au système d'apprendre sur les

connaissances du système, tandis que l'apprentissage profond apprend l'estimation de l'efficacité du système face aux imprévus (la résilience du système). En combinant les deux approches, cet article montre qu'il est possible d'améliorer la résilience du système.

2.4.2 Apprentissage par renforcement inverse

L'apprentissage par renforcement permet à un agent d'optimiser sa stratégie pour s'adapter à son environnement. L'apprentissage par renforcement inverse, présenté par [Ng et Russell, 2000], permet quant à lui de générer une fonction de récompense correspondant au comportement observé d'un autre agent.

Pour cela, l'agent modélise son environnement avec un modèle de markov caché(HMM), introduit par [Eddy, 1996], constitué d'un espace d'états, un ensemble d'actions et une fonction de transition. A partir d'une trajectoire T observée, ou d'une politique π souhaitée, il est possible de construire une fonction de récompense permettant de calculer une politique se rapprochant du comportement souhaité.

Afin de générer cette fonction de récompense, celle-ci est considérée comme étant une combinaison de plusieurs fonction de récompenses. Elle est calculée à partir de l'équation 2.13, où $\phi_i(s)$ est la $i$ième composante de la fonction de récompense, et α_i sa pondération. L'objectif de ce modèle est donc de fixer la valeur des pondérations pour se rapprocher de la politique observée. Pour cela, nous utilisons l'algorithme 4.

$$r_{irl}(s,a) = \sum_{i=1}^{n} \alpha_i \phi_i(s,a) \tag{2.13}$$

Algorithme 4 : Apprentissage par renforcement inverse

Données : Un tuple $\langle S,A,T \rangle$, définissant un MDP sans fonction de récompense;
H trajectoire observée;
Φ ensemble des composantes de la fonction de récompense;

1 $\alpha_i \leftarrow 0 \forall i \in [1..n]$;
2 $k \leftarrow 1$;
3 **tant que** $\langle condition \rangle$ **faire**
4 $\quad k \leftarrow k + 1$;
5 $\quad \forall i \in [1..n]$,$\alpha_i$ mis à jour avec le programme linéaire 2.14;
6 $\quad \pi_k \leftarrow$ politique générée par value iteration (algorithme 2), avec la fonction de récompense r_{irl};

Cet algorithme utilise le programme linéaire décrit dans l'équation 2.14, permettant de maximiser, pour chacune des politiques précédemment calculées, la différence entre l'utilité espérée de celle-ci et l'utilité espérée de la politique observée, renforçant ainsi l'impact de la différence entre la politique optimale et les autres. La fonction p est une fonction doublant les nombre négatifs, afin de diminuer les cas où une politique est plus efficace que l'optimale. Ainsi, il va générer plusieurs politiques, jusqu'à remplir la condition d'arrêt en ligne 3. Selon l'auteur, la condition est d'obtenir une politique dite "satisfaisante". Nous considérons par exemple qu'une politique "satisfaisante" correspond à la trajectoire observée. Néanmoins, il est possible de ne pas trouver de fonction de récompense r_{irl} permettant de générer une politique adéquate. Il est

donc également envisageable de limiter le nombre de politique à un nombre fixe.

$$\begin{array}{ll} \text{Maximiser} & \sum_{i=1}^{k} p(V^{\pi^*}(s_0) - V^{\pi^i}(s_0)) \\ \text{tel que} & |\alpha_i| \leq 1 \forall i \in [1..n] \end{array} \tag{2.14}$$

La méthode d'apprentissage par renforcement inverse présentée par [Ng et Russell, 2000] permet ainsi à un agent de générer une fonction de récompense permettant d'obtenir une politique proche d'une politique observée, dans un environnement observable. [Choi et Kim, 2011] propose une solution pour un environnement partiellement observable. Une autre approche, présentée par [Ziebart *et al.*, 2008], permet d'étudier un ensemble de trajectoires possibles, afin de prendre en compte des trajectoires non-optimales, mais acceptées.

2.5 Planification multi-agents

Plusieurs méthodes de planifications multi-agents existent. Elles répondent toutes à un besoin différent en terme de modélisation. Il est possible de distinguer deux types différents de planification : centralisée et décentralisée.

2.5.1 Planification centralisée

La planification centralisée consiste à calculer la politique de l'ensemble des agents en même temps, puis de l'envoyer aux agents.

Les MMDP (processus decisionnels de markov multi-agent), introduits par [Boutilier *et al.*, 1999], permettent calculer la stratégie de plusieurs agents dans un environnement totalement observable, pour un système coopératif. Un MMDP est défini par un tuple $\langle S,A,T,r \rangle$, où :

- $S = S_1 \times \ldots \times S_n$ est l'espace d'état joint, composé de l'espace d'état de chaque agent.
- $A = A_1 \times \ldots \times A_n$ est l'ensemble des actions jointes des n agents du système
- $T : S \times A \times S \rightarrow [0..1]$ est la fonction de transition, ne dépendant pas d'une action individuelle, mais d'une action jointe $a \in A$.
- $r : S \times A \times S \rightarrow [0..1]$ est la fonction de récompense jointe.

Résoudre un MMDP revient à résoudre un MDP, où les actions sont des actions jointes et les états des états joints. La politique à calculer est alors une politique jointe $\pi = \{\pi_i, \ldots, \pi_n\}$, où π_i est la politique de l'agent i.

Le souci de cette méthode est qu'elle implique lors de l'exécution une connaissance globale de l'état de l'ensemble des agents dans l'environnement, ce qui est rarement vérifié. Il faut alors que les agents communiquent entre eux leur état, ou qu'ils le communiquent à un serveur centralisé, qui leur retournera la prochaine action à effectuer, afin de maintenir l'observabilité totale du modèle.

Un autre problème vient du fait que cette solution implique de calculer la politique de tous les agents. Or, dans le cadre d'un système ouvert, tous les agents ne sont pas forcément connus. De plus, nous considérons dans cette thèse une hétérogénéité des agents en terme d'autonomie, c'est à dire que des agents autonomes évolueront avec des agents télé-opérés, ce qui implique que tous les agents ne verront pas leur politique calculée. Dans ces conditions, il est nécessaire de se tourner vers des solutions décentralisées.

2.5.2 Planification décentralisée

La planification décentralisée consiste à faire planifier à chaque agent sa propre politique, en prenant en compte l'ensemble du système. En pratique, cela revient, dans le monde des MDP, à

en résoudre plusieurs à la fois. Chaque agent a une fonction de récompense qui lui est propre. Plusieurs architectures existent pour calculer l'ensemble des politiques, nous en présentons quelques unes ci-dessous.

Leader-suiveur

L'architecture leader-suiveur est composée d'agents dits leaders, et d'autres dits suiveurs. Le principe de cette architecture est que les leaders décident en premier, puis les suiveurs adaptent leurs stratégies en fonction de celle des leaders. Cette architecture est utilisée dans plusieurs domaines. Par exemple, [Panagou et Kumar, 2014] l'utilise pour maintenir une formation lors d'un parcours d'obstacle, et [Vorobeychik et Singh, 2012] présente un problème de patrouille basé sur le leader-suiveur.

Dans ce modèle, le suiveur calculant sa politique après le leader, c'est lui qui s'adaptera au leader, sa politique est donc calculée à partir de celle du leader. Nous considérons alors un MMDP étendu $\langle S,A,\Pi,T,R\rangle$, où :

— S est l'espace d'état joint, composé de l'espace d'état de chaque agent.
— $A = A_1, \times \ldots \times A_k$ est l'ensemble des actions jointes des k agents suiveurs,
— $\Pi = \{\pi_1, \ldots, \pi_l\}$ est l'ensemble des politiques des agents leaders,
— $T : S \times A \times S'$ est la fonction de transition.
— $R = r_1, \ldots, r_{k+l}$ est l'ensemble des fonctions de récompense pour chaque agent.

Le problème peut se résoudre comme un MMDP, excepté que certaines politiques sont fixées, réduisant fortement le temps de calcul de la politique. Il est également possible d'utiliser d'autres méthodes de résolution décentralisées.

L'avantage de cette méthode est de permettre de considérer des agents en dehors du système de planification, comme par exemple des agents pilotés, à condition d'avoir une connaissance, ou une estimation de leur politique.

Joint Equilibrium-based Search for Policies

Joint Equilibrium-based Search for Policies (JESP), proposé par [Nair *et al.*, 2003], est un algorithme permettant de calculer la politique des agents séparément, de façon itérative. Celui-ci est décrit par l'algorithme 5.

Cet algorithme commence par initialiser, en ligne 1, la politique des n agents. Par la suite, la boucle en ligne 5, permet à chaque agent de mettre à jour sa politique en fonction de la politique jointe actuelle des autres agents en ligne 6. Cette boucle étant recommencée à chaque fois qu'une politique est mise à jour, celle-ci s'arrête lorsque aucun agent n'a d'intérêt à changer sa politique vis à vis de celle des autres. Autrement dit, tous les agents ont trouvé un équilibre de Nash.

Définition 8. *Un équilibre de Nash est une situation de décision dans laquelle aucun agent n'a d'intérêt à changer de décision, compte tenu de la décision des autres agents.*

L'avantage de cette solution est de diminuer de manière significative le temps de calcul d'une solution. Elle permet, ainsi, de traîter un nombre d'agents plus importants, et de faire en sorte que chacun d'entre eux prennent en compte les actions des autres. Son inconvénient est que celle-ci converge vers un optimum local. Ainsi, les agents n'explorant pas toutes les solutions, il est possible qu'une combinaison de politiques soit plus efficace que celle trouvée, mais ne soit pas atteinte durant l'exécution de l'algorithme de planification, car plusieurs agents devraient changer de politique en même temps pour améliorer leurs espérances respectives.

Algorithme 5 : JESP

Données : MMDP$\langle S,A,T,R\rangle$, nombre d'agents n

`/* Initialisation */`

pour chaque $i \in [1..n]$ **faire**
  pour chaque $s_t \in S$ **faire**
    $\pi_i(s)$ défini arbitrairement;
$i \leftarrow 1$;
tant que $i \leq n$ **faire**
  $\pi'_i \leftarrow$ maximisation($\pi_m, m \in [1..n], m \neq i$);
  si $\pi_i \neq \pi'_i$ **alors**
    $\pi_i \leftarrow \pi'_i$;
    $i \leftarrow 1$
  sinon
    $i \leftarrow i + 1$;
retourner $\pi_i \forall i \in [1..n]$

2.5.3 DEC-POMDP

Comme dans le cadre mono-agent, un système multi-agent peut difficilement assurer une observabilité totale du système. Un Processus Décisionnel de Markov Partiellement Observable Décentralisé (DEC-POMDP), présenté par [Bernstein *et al.*, 2000], est un modèle permettant de définir un système multi-agent décentralisé dans un environnement partiellement observable. Il est défini par un tuple $\langle S,A,T,r,\Omega,O\rangle$, où :

- $S = S_1 \times \ldots \times S_n$ est l'espace d'état joint, composé de l'espace d'état de chaque agent.
- $A = A_1 \times \ldots \times A_n$ est l'ensemble des actions jointes des n agents du système
- $T : S \times A \times S \rightarrow [0..1]$ est la fonction de transition, ne dépendant pas d'une action individuelle, mais d'une action jointe $a \in A$.
- $r : S \times A \times S \rightarrow [0..1]$ est la fonction de récompense jointe.
- $\Omega : \Omega_1 \times \ldots \Omega_n$ est l'ensemble joint des observations que chaque agent peut avoir sur son environnement
- $O : S \times AS \times \Omega \rightarrow [0..1]$ est la fonction d'observation jointe du système.

Résoudre un DEC-POMDP est très difficile, car il faut prendre en compte un grand système, vu au nombre d'agents, ainsi que l'observabilité partielle. De nombreuses astuces ont été proposées pour simplifier la résolution du système, comme par exemple la factorisation des états ([Bernstein *et al.*, 2000]), ou la séparation des agents en cluster d'agents pouvant s'influencer ([Canu, 2011]).

2.5.4 Application à un système à autonomie hétérogène

JESP permettrait ainsi à notre système de calculer une politique pour les agents autonomes de notre système entre eux. Elle permet également de considérer d'autres agents, hors du système de planification, en considérant leur politique fixée. Cela nous permet donc d'inclure dans la planification des agents qui seraient pilotés. Néanmoins, cela implique de connaître leur politique, alors assimilable à celle de l'opérateur. A moins que celui-ci ne la communique directement, il va falloir l'estimer.

2.6 Planification à initiative mixte

Il arrive parfois qu'au cours de la mission d'un agent, certaines tâches puissent être résolues facilement sans intervention de l'opérateur, mais que d'autres tâches requièrent le savoir faire d'un opérateur. Prenons par exemple le cas de la navigation en zone accidentée. Dans une zone moins accidentée, le robot est en mesure d'évoluer de manière autonome, sans intervention de l'opérateur. Néanmoins, s'il approche d'une zone accidentée, la perception de l'opérateur peut être requise pour éviter au robot une chute, ou une collision. Il peut alors être intéressant de permettre à l'opérateur de reprendre le contrôle du robot dans ce genre de situation.

[Horvitz, 1999] présente un problème de planification à initiative mixte où un agent a le contrôle sur un système, mais partage ce contrôle avec un opérateur. Par exemple, l'agent peut prendre le contrôle du système pour des tâches simples, mais l'opérateur reprend l'initiative des actions pour des tâches dont les résolutions sont moins certaines. [Mouaddib *et al.*, 2010b] propose par exemple de permettre à un robot de naviguer dans son environnement de manière autonome. Néanmoins, celui-ci a une fonction de transition différente, voire moins précise que celle de l'opérateur. Il peut alors arriver des situations complexes où il est plus intéressant de demander à l'opérateur de prendre la main plutôt qu'essayer de maintenir l'autonomie, comme par exemple dans un couloir étroit, où le robot peut se prendre un mur. [Ferguson *et al.*, 1996] propose quant à lui TRAINS-95, une application d'assistant pour la planification dans un système de routage pour des transports.

[Horvitz, 1999] présente différents principes pour la mise en place d'une interface pour un système à initiative mixte, comme par exemple la prise en compte de l'incertitude quant à l'attention ou le but de l'opérateur, ou l'emploi de dialogue pour clarifier les incertitudes, comme l'attention de l'utilisateur.

2.7 Conclusion

Nous avons montré dans ce chapitre comment modéliser un problème de planification à l'aide d'un espace d'états, une liste d'actions, une fonction de transition et un critère d'évaluation basé sur une somme de gains et de coûts dans un environnement déterministe.

Nous avons ensuite appliqué cette méthode sur l'exemple de navigation d'un robot dans un bâtiment sécurisé, présenté en section 1.2.2. Enfin, nous avons proposé une résolution de cette méthode basée sur la maximisation des gains de l'agent.

Nous avons introduit plusieurs outils de planification tels que les Processus Décisionnels de Markov, permettant à un agent de modéliser un environnement stochastique et d'y calculer sa stratégie, sous forme d'une politique. Nous avons aussi présenté un exemple permettant d'illustrer la modélisation avec un MDP. Nous avons également introduit deux méthodes de résolution des MDP, Value Iteration et Policy Iteration, et présenté leurs différences.

Nous avons ensuite introduit les Processus Décisionnels de Markov Partiellement Observables (POMDP), qui permettent de planifier une politique dans un environnement partiellement observable. Nous avons notamment expliqué que ce modèle était plus coûteux qu'un MDP, que ce soit en terme de mémoire ou de calcul de la politique.

Nous avons ensuite parlé de la planification multi-agents, et présenté deux types de fonctionnement. Le premier, centralisé, permet de planifier de manière coordonnée les actions de l'ensemble des agents en même temps. Le second, décentralisée, permet de planifier les actions des agents séparément, en les adaptant aux politiques déjà générés pour les autres agents. Outre le fait que la seconde solution est moins couteuse, celle-ci semble plus adaptée à un environne-

ment où des agents peuvent devenir pilotés, dans la mesure où la politique d'un agent piloté ne peut être générée, puisque celui-ci est directement contrôlé par l'opérateur.

Nous avons ensuite parlé des limites des systèmes autonomes, comme leur incapacité à s'adapter à un imprévu, et ce bien que nous considérions dans cette thèse que les agents sont en mesure de capter les principales informations de leur environnement et de s'y repérer totalement. De plus, l'opérateur peut détenir des informations que Nous avons alors présenté la planification à initiative mixte, permettant de partager le contrôle du robot avec l'agent et l'opérateur. Ainsi, l'agent peut agir de manière autonome pour les tâches simples, et confier les tâches compliquées, comme la navigation au bord d'un précipice, à l'opérateur.

Ceux-ci seront capables à la fois d'imposer des restrictions aux agents, d'interdire le déplacement vers une zone, de prendre le contrôle de l'agent au cours de l'exécution. Ainsi, l'autonomie de l'agent n'est pas totale, on parle donc de **semi-autonomie**.

Nous expliquerons dans le chapitre 3 ce qu'est la semi-autonomie, ainsi que différents niveaux d'autonomies. Nous parlerons également d'approches existantes pour permettre à l'opérateur d'agir sur la politique de l'agent.

Chapitre 3

Semi-Autonomie

Sommaire

L'autonomie est une question importante dans le recherche en intelligence artificelle. Elle définit la capacité d'un agent à se passer de l'intervention d'un opérateur. Ainsi, sans autonomie, afin de faire agir l'agent, il est nécessaire de le piloter. Néanmoins, certaines tâches sont automatisables, et permettre à un agent de s'occuper de celles-ci permet à l'opérateur de s'attarder sur des tâches plus importantes. Un agent contrôlant toutes les actions d'une entité (robot, programme, ...) est dit autonome.

Seulement, l'autonomie complète n'est pas toujours facile à mettre en place. En effet, le robot n'est pas à l'abri de défauts de capteurs, diminuant la capacité du robot à observer son environnement et donc d'en déduire son état. Il peut également se retrouver dans une situation non prévue, dans le cadre d'un environnement ouvert, remettant en cause le plan calculé auparavant. Pour faire face à cette difficulté, des compromis ont été trouvés, impliquant l'opérateur dans la mission, mais de façon moins importante que par le pilotage.

Dans ce chapitre, nous parlerons dans un premier temps de la semi-autonomie, puis de l'autonomie ajustable.

3.1 Définitions

On parle de semi-autonomie lorsque l'agent dépend partiellement de son opérateur. On y distingue en général plusieurs niveaux, en fonction de l'implication de l'opérateur dans la mission de l'agent. Plusieurs visions de la semi-autonomie existent, nous parlerons ici de deux d'entre elles : la semi-autonomie à l'exécution, et à la planification.

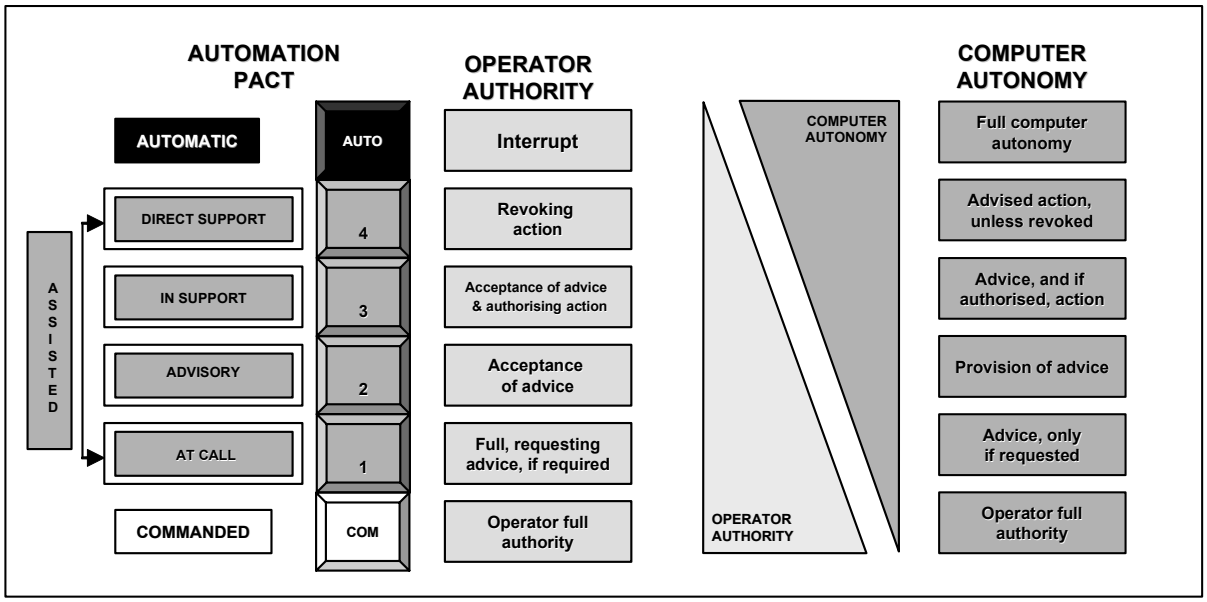


FIGURE 3.1 – Graphique tiré de [Taylor, 2003], présentant P.A.C.T.

3.1.1 Semi-autonomie à l'exécution

La semi-autonomie au cours de l'exécution consiste à évaluer qui, entre l'agent et l'opérateur, contrôle le système. Nous pouvons prendre pour exemple un avion, avec pilotage automatique. Au cours de sa mission de vol, le pilote peut choisir de laisser le système naviguer de manière autonome, ou au contraire de prendre le contrôle.

La figure 3.1, présentée par [Taylor, 2003], présente le framework **P.A.C.T.** (Pilot authority and control of tasks), permettant de définir des niveaux d'autonomie. Ce framework définit le niveau d'autonomie en fonction de l'autorité de l'agent et de l'opérateur sur le système. Entre autres, on y note que cette autorité évolue de manière symétrique en fonction du niveau d'autonomie.

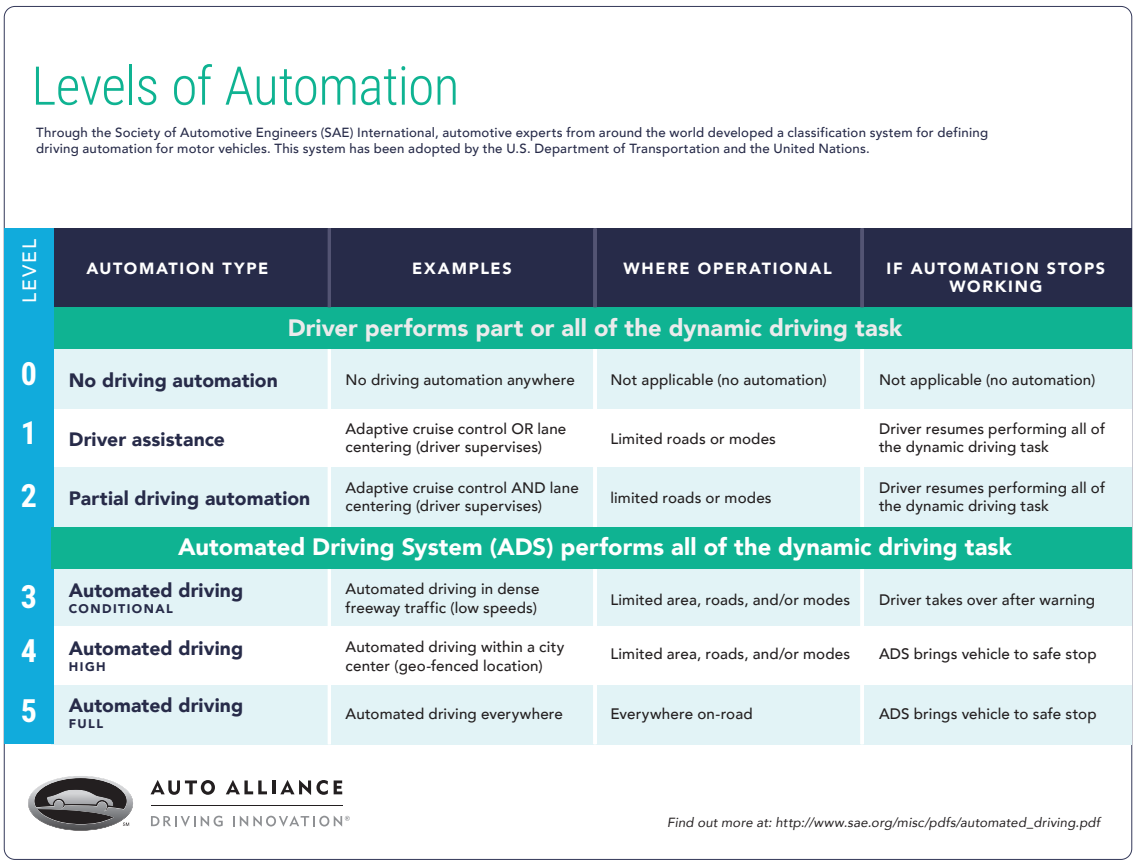
Au niveau 0, l'agent est piloté, et au niveau 5, l'agent est autonome. Entre les deux, l'agent est "assisté", ou semi-autonome. On y distingue plusieurs niveaux d'autonomie, en fonction des interactions possibles entre l'opérateur et l'agent. Par exemple, un opérateur peut avoir pour rôle de surveiller l'agent, ou l'agent peut juste être un conseiller pour l'opérateur.

En général, peu importe le niveau d'autonomie de l'agent, sa stratégie est calculée de la même façon. Les différents niveaux d'autonomies permettent de choisir quel plan, entre celui de l'agent et celui de l'opérateur, sera appliqué, ou alors lequel des deux appliquera le plan. Au final, il est surtout question d'autorité, de confiance, de responsabilité ou d'efficacité à l'exécution.

D'autres systèmes de classifications de système semi-autonomes existent, tels que celui du Society of Automotive Engineers Internationnal (SAE-I) ([International, 2016]) utilisé pour les systèmes de pilotage automatique, présenté en figure 3.2 [2].

Cette classification suggère soit que le système est entièrement piloté (niveau 0), soit qu'il ne fait que de légères corrections de trajectoire , mais nécessitent l'attention de l'opérateur (niveaux 1 et 2). Le système peut être également autonome de manière situationnelle (niveaux 3 à 5). Il suggère également comment l'opérateur doit reprendre la main en cas d'arrêt du pilotage automatique.

2. https://autoalliance.org/wp-content/uploads/2017/07/Automated-Vehicles-Levels-of-Automation.pdf



Levels of Automation

Through the Society of Automotive Engineers (SAE) International, automotive experts from around the world developed a classification system for defining driving automation for motor vehicles. This system has been adopted by the U.S. Department of Transportation and the United Nations.

LEVEL	AUTOMATION TYPE	EXAMPLES	WHERE OPERATIONAL	IF AUTOMATION STOPS WORKING
	Driver performs part or all of the dynamic driving task			
0	**No driving automation**	No driving automation anywhere	Not applicable (no automation)	Not applicable (no automation)
1	**Driver assistance**	Adaptive cruise control OR lane centering (driver supervises)	Limited roads or modes	Driver resumes performing all of the dynamic driving task
2	**Partial driving automation**	Adaptive cruise control AND lane centering (driver supervises)	limited roads or modes	Driver resumes performing all of the dynamic driving task
	Automated Driving System (ADS) performs all of the dynamic driving task			
3	**Automated driving** CONDITIONAL	Automated driving in dense freeway traffic (low speeds)	Limited area, roads, and/or modes	Driver takes over after warning
4	**Automated driving** HIGH	Automated driving within a city center (geo-fenced location)	Limited area, roads, and/or modes	ADS brings vehicle to safe stop
5	**Automated driving** FULL	Automated driving everywhere	Everywhere on-road	ADS brings vehicle to safe stop

AUTO ALLIANCE
DRIVING INNOVATION®

Find out more at: http://www.sae.org/misc/pdfs/automated_driving.pdf

FIGURE 3.2 – Niveaux de semi-autonomie selon SAE-I

3.1.2 Semi-autonomie à la planification

La semi-autonomie au cours de la planification permet à un opérateur d'interférer avec le plan initial construit à l'origine par l'opérateur. En effet, un agent totalement autonome calcule une politique en fonction de son objectif, et en fonction du coût de ses actions. Il sera souvent question d'ajouter des contraintes au cours de l'exécution.

Une solution est d'intégrer directement ces contraintes dans le modèle de planification. [Mouaddib *et al.*, 2015] proposent, par exemple, une extension des MDP pour permettre à un opérateur de définir des objectifs, des états indésirables, des états interdits et des états souhaités. L'agent doit alors calculer une politique respectant des contraintes, comme éviter à tout prix les états interdits, et si possible les états indésirables.

Une autre méthode, proposée par [Sprauel *et al.*, 2014] consiste en une extension des MDP visant à rajouter d'autres types de contraintes. Par exemple, celle d'atteindre un état avant un autre, ou de passer un certain nombre de fois par un état. On retrouve également des contraintes visant à assurer un pourcentage de succès pour atteindre un état.

En général, ces méthodes impliquent l'opérateur lors de la préparation de la mission, mais permettent à l'agent d'être autonome à l'exécution de celle-ci.

Bien que la semi-autonomie est visible sur ces deux plans, il est possible de combiner les deux visions. Par exemple, une politique générée à partir de contraintes pourrait nécessiter

l'approbation de l'opérateur malgré tout.

Malgré tout, il est parfois possible qu'un niveau bien défini ne suffise pas aux besoins de l'opérateur. Par exemple, il est possible que pour certaines tâches, l'opérateur soit nécessaire et pour d'autres non. Pour cela, il pourrait être intéressant de pouvoir faire varier le niveau de l'autonomie de l'agent au cours de la mission.

3.1.3 Types de systèmes semi-autonomes

[Zilberstein, 2015] propose une méthode pour catégoriser les différents systèmes autonomes, en fonction de leur robustesse face aux imprévus :

- Un système semi-autonome de type I (SAS-I) ne contient pas l'opérateur dans son modèle. Ainsi, l'agent n'est pas en mesure d'agir en fonction de ce que l'opérateur peut faire, mais ce dernier peut interférer avec la mission de l'agent pour l'aider.
- Un système semi-autonome de type II (SAS-II) inclue les interventions possibles de l'humain dans son modèle. Il est ainsi en mesure de lui demander des informations sur son environnement, de prendre le contrôle.

De manière générale, un système de type SAS-II est en mesure de demander de l'aide à l'opérateur avant d'arriver dans une situation de *dead-end*, où il ne serait plus en mesure de lui répondre. Ce n'est pas le cas du système de type SAS-I, puisque celui-ci n'inclut pas l'opérateur dans son modèle.

Enfin, [Zilberstein, 2015] définit un *strong SAS*, par opposition au *weak SAS*, comme étant un système de type SAS-II qui s'assure de ne pas finir en *dead-end*. Il est donc en mesure d'assurer son fonctionnement tout en étant capable de demander de l'aide à un opérateur, si nécessaire.

3.1.4 Autonomie Ajustable

L'autonomie ajustable est une catégorie de la semi-autonomie dans laquelle l'autonomie de l'agent est amenée à évoluer dans la mission. L'opérateur peut par exemple choisir de reprendre la main sur le robot lorsque celui-ci est en difficulté ou fait face à un imprévu. Elle permet à un agent dont certaines tâches sont simples de les accomplir de manière autonome, tandis que l'opérateur interviendra pour les tâches plus compliquées. Le temps libéré pour l'opérateur lui permettra ainsi, par exemple, de superviser d'autres agents.

Autonomie adaptative

D'après [Côté, 2013], l'autonomie adaptative est un cas particulier de l'autonomie ajustable. Il s'agit des agents qui sont en mesure de demander eux-même le changement de niveau d'autonomie à l'opérateur. Il correspond donc au moins à un système semi-autonome de type SAS-II.

D'après [Baker et Yanco, 2004], bien qu'une gestion de l'autonomie de l'agent par l'opérateur peut être plus efficace en terme d'interactions, il semblerait qu'en pratique, ce soit rarement l'opérateur qui change le niveau d'autonomie de l'agent. Elle suggère d'ailleurs, pour changer le niveau d'autonomie, de laisser l'agent demander à l'opérateur le changement d'autonomie. Cela permet à l'agent d'agir en attendant que l'opérateur se prépare à reprendre la main. L'opérateur peut également faire autre chose en attendant, plutôt que de guetter une situation où il devrait reprendre la main, comme par exemple s'occuper d'autres agents.

L'autonomie adaptative n'est pas souvent dissociée de l'autonomie ajustable dans l'état de l'art, nous ne les distinguerons donc plus dans la suite du document.

Transfert de contrôle

Qui dit autonomie ajustable dit transfert de contrôle. Comme dit précédemment, ce sera en général l'agent qui annoncera la demande de prise de contrôle. Nous pouvons distinguer plusieurs raisons pour l'agent de demander une prise de contrôle, dont celles ci-dessous :

1. Lorsque l'impact de ses actions baisse, et donc son utilité, [Crandall *et al.*, 2005] propose de faire appel à l'opérateur pour aider le robot en lui donnant de nouvelles directives.
2. [Kennedy, 1999] propose de faire appel à l'opérateur lorsque l'agent détecte une anomalie dans le robot, par exemple au niveau de ses capteurs. L'opérateur peut en effet disposer d'autres informations permettant de compenser les anomalies du robot.
3. [Mouaddib *et al.*, 2010a] propose d'évaluer les capacités de l'opérateur à accomplir la tâche de l'agent à sa place. Si l'opérateur est significativement plus efficace, au point qu'il soit plus intéressant de déranger l'opérateur, alors l'agent demande à l'opérateur de prendre le contrôle. Ce genre de comportement nécessite néanmoins de posséder un modèle de planification pour l'opérateur. L'interaction mixte, présenté en section 2.6, est une solution pour que l'agent puisse prendre ce genre de décisions. [Carlson et Demiris, 2008], [Fong *et al.*, 2001] et [Rosenthal *et al.*, 2012] fonctionnent également sur le principe où l'agent choisit qui de l'opérateur ou lui-même sera le plus efficace.
4. [Armstrong-Crews et Veloso, 2007] et [Rosenthal et Veloso, 2011] proposent quant à eux de permettre à l'agent de demander des précisions sur son état à un opérateur dans un environnement partiellement observable.
5. [Lopes *et al.*, 2009] et [Cohn *et al.*, 2011] proposent quant à eux de permettre aux agents de demander à l'opérateur quelle action choisir pour apprendre par démonstration une fonction de récompense.

Partage d'autorité

Une autre approche utilisée dans l'autonomie ajustable est le partage d'autorité. [Mercier, 2011] propose un modèle permettant à un agent de partager l'autorité des agents. Elle permet, par exemple, d'assigner la responsabilité d'une fonction d'un système à un agent pour qu'il assure son fonctionnement dans un système multi-agent. Ainsi, l'agent peut refuser à un autre l'accès à une ressource du système s'il estime qu'il enfreint ses priorités, tant qu'il en a l'autorité. Par exemple, l'agent peut refuser à un pilote d'emmener le robot trop loin s'il risque d'épuiser toute la batterie du robot.

C'est une approche intéressante car elle suppose que l'agent a la capacité d'interdire des comportements qui pourraient nuire aux priorités de la mission. Néanmoins, cette approche est une solution à un autre problème où l'opérateur n'a pas assimilé toutes les informations utiles pour la prise de décision.

Dans notre cas, nous souhaitons justement nous appuyer sur les informations que l'opérateur a en plus du système pour améliorer le comportement du robot, il doit donc toujours avoir la capacité de reprendre le contrôle sur le système.

Applications de l'autonomie ajustable

L'autonomie ajustable est un domaine très actif dans la recherche. Elle fut d'abord proposée par [Dorais *et al.*, 1999] et [Bradshaw *et al.*, 2003] dans le cadre des missions spatiales. Depuis,

elle est de plus en plus utilisée pour des domaines plus courant, comme les maisons intelligentes ([Lesser *et al.*, 1999]), le commerce électronique ([Collins *et al.*, 2000]) ou l'interaction avec les drones ([Clough, 2002]). Ci-dessous, nous décrivons plusieurs exemples d'applications d'autonomie ajustable.

Mixed-Initiative MDP [Mouaddib *et al.*, 2010a] propose une extension de MDP permettant à un agent d'évaluer les capacités d'un opérateur à accomplir son objectif à partir de fonctions de transitions différentes en exploitant le concept d'initiative mixte. L'agent peut ainsi demander à l'opérateur de prendre le contrôle du robot s'il considère l'opérateur plus efficace que lui. Il faut pour cela intégrer un modèle de l'opérateur dans le MDP.

Ainsi, l'espace d'état est augmenté pour prendre en compte le niveau d'attention de l'opérateur. L'agent dispose d'actions supplémentaires, visant à augmenter, ou diminuer le niveau d'attention qu'il attend de l'opérateur. Ces actions permettent de modifier progressivement le niveau d'attention attendu de l'opérateur, lui laissant ainsi le temps de comprendre la situation.

L'opérateur quant à lui dispose lui aussi des actions permettant de contrôler le robot, mais également des actions permettant de passer d'un niveau d'autonomie à un autre.

Pour modéliser l'efficacité de l'opérateur, différente de celle de l'agent, [Mouaddib *et al.*, 2010a] propose de considérer une fonction de transition différente entre celle de l'opérateur et celle de l'agent. Un coût est également ajouté à la fonction d'utilité en fonction du niveau d'attention de l'opérateur, afin de maximiser le fonctionnement en autonomie du système.

Cette méthode permet ainsi de modéliser la prise de contrôle de l'opérateur dans le système, en considérant qu'il soit plus ou moins efficace par rapport à l'agent. Il permet également de planifier le transfert de contrôle du robot entre l'humain et l'agent en fonction de leur efficacité et du coût de ce transfert. Néanmoins, ce modèle suppose de connaître la fonction de transition de l'opérateur, ce qui parait compliqué à assurer.

Partage d'autorité [Côté, 2013] a présenté au cours de sa thèse un travail permettant à un opérateur de transmettre des recommandations à un opérateur, que ce soit, par exemple, pour l'inciter à atteindre par un état ou pour l'éviter. Cela permet à l'opérateur d'influencer la politique de l'agent sans pour autant monopoliser son attention sur une longue période. Ces recommandations positives sont intégrées en modifiant l'utilité associée aux états à atteindre. Pour les états à éviter, la fonction de transition est modifiée de sorte à ce que l'agent ne puisse plus sortir de cet état lors de la planification.

Néanmoins, ces recommandations peuvent influencer positivement, ou négativement la résolution de la mission en cours. Pour éviter des impacts négatifs répétés sur l'accomplissement de l'objectif, [Côté, 2013] propose d'intégrer une notion de confiance dans les recommandations de l'opérateur. En fonction de cette confiance, et de la perte d'utilité associée à cette recommandation, l'opérateur va plus ou moins suivre la recommandation de l'opérateur. Ainsi, l'autonomie ajustable ne fait pas référence au contrôle, mais sur la prise de décision quant à l'intégration des recommandations de l'opérateur.

Autonomie ajustable pour planification de rendez-vous [Scerri *et al.*, 2002] a mis en place Electric Elves, un projet de déploiement d'un agent pour organiser les activités journalières de personnes, comme les rendez-vous. Il fonctionnait sur un système où chaque personne est connectée à un agent appelé *Friday*, qui communique avec l'ensemble des autres *Friday* en broadcast. Chaque Friday implémentait le framework *STEAM*, présenté par [Tambe, 1997]. Les Friday permettaient aux utilisateurs de planifier leurs réunions, et de déterminer, en cas de

retard, s'il était plus intéressant de confier l'animation de la réunion à une autre personne, ou de reporter la réunion, afin de minimiser les pertes de temps des utilisateurs ou leur perturbation.

Ce projet était l'occasion de tester l'autonomie ajustable en conditions réels sur des sujets peu abordés. Par exemple, il y est question de l'autonomie ajustable dans un système multi-agents. En effet, lorsque l'agent demande à l'utilisateur si une réunion doit être reporté, ou annulé, le temps de réponse de la personne peut entraîner une désynchronisation du système, où les personnes vont au rendez-vous alors que la personne animant la réunion est absente.

Pour assurer la cohésion du système, celui-ci prend des décisions en respectant différentes contraintes, comme s'assurer que tous, ou personne ne soit à la réunion. Il peut également proposer à quelqu'un d'autre de remplacer un absent à la réunion, à condition qu'il soit disponible, ait les compétences et le temps pour préparer la réunion.

Ce projet a permis d'obtenir de bonnes avancées dans la recherche. Néanmoins, celui-ci a été arrêté, selon [Tambe, 2008], notamment à cause de problème de respect de la vie privée.

3.2 Evaluation

3.2.1 Bonnes pratiques

[Zilberstein, 2015] présente un état de l'art des systèmes semi-autonomes. Il y décrit entre autres les pratiques attendues favorisant la robustesse d'un système semi-autonome. On y retrouve par exemple la nécessité d'éviter les impasses, ou les états qui, une fois atteints, empêchent d'accomplir la mission ou d'interagir avec l'opérateur. L'agent doit aussi inclure dans son modèle les interactions possibles avec l'opérateur, et être en mesure de détecter ses intentions et de prendre en compte la notion du temps de l'humain pour modéliser son état. En effet, la décision d'un agent est cadencée par son processeur, où l'humain doit d'abord percevoir l'information de l'agent, puis réfléchir avant de répondre. Les décisions et les actions de l'opérateur ne se feront donc pas sur la même échelle temporelle que celle de l'agent. Enfin, il est nécessaire, pour l'autonomie ajustable, que le transfert de contrôle soit rapide, mais qu'il assure aussi pour l'opérateur de saisir l'état actuel de l'agent dans son environnement avant la prise de contrôle.

3.2.2 Critères d'évaluation

Afin d'évaluer l'efficacité d'un système semi-autonome, différents critères sont envisageables, en fonction de ce que l'on souhaite représenter. Nous en donnons quelques exemples ci-dessous

Coût d'interruption

Lorsqu'un agent semi-autonome est en cours de mission, il est lui arrivera d'interagir avec l'opérateur. Seulement, toutes les interactions n'ont pas le même impact sur la mission de l'opérateur. Par exemple, autoriser une action à un agent prend moins de temps à l'opérateur que de le télé-opérer.

Pour représenter l'impact des interactions sur l'opérateur, [Fleming et Cohen, 2004] définit le coût d'interruption.

Définition 9. *Le coût d'interruption (ou bother-cost) est le fréquence à laquelle un agent sollicite un opérateur.*

D'après [Bailey *et al.*, 2001], ces interruptions sont sources d'anxiété pour les opérateurs et peuvent nuire à l'efficacité de l'agent et de l'opérateur au cours de la mission.

Ce coût d'interruption peut être modélisé dans la fonction d'utilité de l'agent, lui permettant non seulement de choisir une interaction appropriée au besoin de l'agent, mais aussi d'assurer que l'apport de cette interruption soit réel pour l'agent.

Temps de négligence

Le temps de négligence, défini par [Crandall *et al.*, 2005] est un critère utilisé dans les systèmes où sans intervention d'un opérateur, un agent perd de l'utilité au cours de la mission.

Par exemple, imaginons un système où un opérateur envoie un robot en reconnaissance dans une pièce. Celui-ci gagne de l'utilité pour chaque nouveau coin découvert dans la pièce, et ce jusqu'à ce que la pièce soit sécurisée, ou qu'une menace soit détectée. Lorsque l'inspection de la pièce est terminée, l'agent a besoin que l'opérateur lui désigne une nouvelle pièce à inspecter pour regagner de l'utilité.

L'agent est alors supposé demander de l'aide à l'opérateur lorsque son utilité atteint un seuil minimum de performance, afin que l'intervention de l'opérateur permette d'augmenter l'efficacité du robot. Ici, le temps de négligence est le temps que met l'agent à nécessiter l'aide de l'opérateur. Ainsi, plus le temps de négligence est faible et moins l'agent est autonome.

[Crandall *et al.*, 2005] propose également de mettre à profit le temps où l'opérateur n'interagit pas avec cet agent pour interagir avec d'autres. Elle propose ainsi de calculer le **fanout**, défini par [Olsen et Goodrich, 2003], correspondant au nombre d'agents que l'opérateur peut superviser en même temps. Celui-ci est défini telle que décrit dans l'équation 3.1, où NT est le temps de négligence, et IT est le temps où l'opérateur est sollicité.

$$fanout = \frac{NT}{IT} + 1 = \frac{NT + IT}{IT} \tag{3.1}$$

Le temps de négligence permet ainsi d'avoir des informations sur le temps d'occupation d'un opérateur, et le nombre de robots supervisables par opérateur.

3.3 Conclusion

Nous avons parlé dans ce chapitre des différents niveaux d'autonomies. Nous avons également défini la semi-autonomie, et parlé de deux visions de celle-ci et de leurs intérêts. Ainsi, la semi-autonomie au cours de l'exécution permet de définir l'autorité/la responsabilité en fonction des capacités de l'agent ou des conséquences de ses actions, alors que la semi-autonomie permet à l'agent d'inclure des contraintes au cours de la planification.

Nous avons également parlé de la nécessité d'ajuster en temps réel l'autonomie de l'agent, en fonction de la difficulté de la tâche.

Enfin, nous avons expliqué quels critères étaient utilisés pour juger d'un bon système semi-autonome, et présenté quelques critères d'évaluation, comme le temps de négligence, permettant de calculer le nombre d'agents qu'un opérateur peut encadrer, ou le coût de dérangement, visant à évaluer l'autonomie de l'agent.

Conclusion de la Partie 1

L'objectif de cette thèse est de permettre de nouveaux types d'interactions entre agents à autonomie ajustable et opérateurs. D'une part, nous souhaitons permettre à des opérateurs de définir des restrictions sur l'autonomie de l'agent. Plusieurs solutions existent, permettant à un agent de demander de l'aide à l'opérateur, lorsqu'il est en difficulté. Néanmoins, ceux-ci nécessitent à un agent d'être non seulement capable de déterminer quand il a besoin de l'aide de l'opérateur, mais aussi de déterminer si celle-ci sera utile. Il lui faut donc un modèle de l'humain pour connaître ses capacités, ce qui, en plus d'être complexe, peut varier pour chaque personne.

De plus, il existe des situation où l'opérateur peut décider lui-même les situations où il doit intervenir. Par exemple, un robot équipé d'une arme à feu ne peut tirer de lui même sur une cible sans autorisation, afin d'éviter d'abattre un civil. De même, l'opérateur peut vouloir surveiller le passage du robot dans une zone précise, soit car elle est dangereuse, ou qu'il cherche quelque chose dans cette zone que le robot ne peut percevoir directement, mais que l'opérateur est en mesure de la voir à travers la caméra du robot.

D'autre part, nous souhaitons permettre aux agents de se coordonner à un agent dont le niveau d'autonomie a été modifié. Par exemple, lorsqu'un opérateur prend le contrôle de l'agent, les autres agents ne sont plus en mesure de connaître les futures actions du robot, puisque les actions sont faites par l'opérateur. De plus, l'opérateur étant influencé par sa perception et ses connaissances de l'environnement, celui-ci peut piloter de manière non-optimale, selon le point de vue des agents autonomes. Il faut alors que ces agents s'adaptent à l'opérateur pour l'assister.

Avec cet objectif, nous avons dans cette partie exploré l'état de l'art.

Agents, planification et environnement

Dans un premier temps, nous avons défini ce qu'est une intelligence artificielle. Par la suite, nous avons parlé des problèmes de planification, permettant à un agent de décider de manière autonome de quelle façon agir pour accomplir un objectif. Nous avons ensuite parlé des systèmes multi-agents, ainsi que les questions à se poser à leur sujet, comme par exemple si les agents cherchent à s'entraider ou au contraire à s'affronter, ou si le système est homogène, c'est à dire que les agents sont similaires.

Dans notre cas, nous nous intéressons à des agents qui ont une mission en commun, et qui sont en mesure de communiquer entre eux. Nous considérons donc un environnement observable, avec des agents coopératifs. Néanmoins, le système est ouvert, puisqu'il est possible d'avoir des imprévus, d'où l'idée de permettre à un opérateur de reprendre la main sur le robot en cas de difficultés. Les agents ne sont donc pas tous forcément autonome, le système est dont hétérogène.

Planification, décision et apprentissage

Dans un deuxième temps, nous avons présenté plusieurs méthodes pour planifier les action d'un agent avant la mission. La première méthode, proposant de modéliser le problème de planification sous forme d'un espace d'états, d'actions, d'une fonction de transition et d'une fonction d'utilité, n'était pas suffisante, car elle supposait un environnement déterministe, c'est à dire que les conséquences de chaque action étaient connues à l'avance. Hors, les actions d'un robot ont tendance à manquer de précision, ce qui fait qu'il y aura toujours une marge d'erreur entre ce qu'on lui demande et ce qu'il fait réellement. Par exemple, en tentant d'avancer de 10cm, celui-ci peut aisément arriver à 9 ou à 11cm en fonction de la précision de ses moteurs. Pour remédier à ce problème, nous nous sommes tournés vers les Processus Décisionnels de Markov, permettant de calculer une stratégie, ou une politique, à un agent de la même façon, mais dans un environnement stochastique, c'est à dire que le système connait la probabilité qu'a le robot d'arriver dans un état pour chaque action qu'il effectue dans un état donné.

Par la suite, nous avons présenté une méthode permettant à un agent de décider non pas à partir de son état, mais à partir de son observation. En effet, un robot n'est pas si souvent dans la situation où il peut déduire l'état de tout le système partir de ses capteurs. Le modèle se base sur l'hypothèse qu'à partir d'une croyance initiale sur l'état dans lequel on se trouve, et à partir des actions et des observations reçues des capteurs, un agent est en mesure d'en déduire une croyance sur l'état dans lequel il se trouve actuellement. Cette méthode est intéressante pour notre problème, car elle permettrait, par exemple, de s'adapter à un environnement accidenté où ses capteurs ne pourraient lui offrir toutes les informations dont il a besoin. Néanmoins, nous faisons dans cette thèse l'hypothèse que l'agent perçoit son environnement pour se concentrer sur les interactions entre l'opérateur et les agents.

Nous avons ensuite étudié la planification multi-agent et présenté les deux principaux axes visant à calculer une politique pour l'ensemble des agents. La première, dite centralisée, consiste à calculer l'ensemble des politiques de tous les agents afin d'obtenir un comportement coordonné, mais cette approche est lourde en calculs. Une autre solution est la planification décentralisée, permettant au système de calculer itérativement la politiques de chaque agent, en l'adaptant à celle des autres, jusqu'à ce que ces politiques convergent. Dans notre problème, certains robots peuvent être pilotés. Ainsi, l'opérateur a le contrôle et ses actions ne sont alors pas planifiés. C'est pourquoi nous nous tournons vers une approche décentralisée.

Néanmoins, cela implique d'avoir connaissance, ou une estimation de la politique que va exécuter le pilote du robot, afin de se coordonner avec lui. Hors, ce pilote étant humain, celui-ci a une perception et des connaissances différentes des robots. Notamment, du fait de ses connaissances, il peut avoir une appréciation différente des états du système. Son comportement peut donc être perçu comme non optimal pour les autres agents, rendant difficile l'estimation de sa politique.

Pour aborder la question de l'estimation de la politique d'un opérateur, nous nous sommes intéressé à l'apprentissage, permettant à un agent d'apprendre le modèle de planification au cours de la mission. Nous avons évoqué l'apprentissage supervisé permettant, à un agent d'apprendre les actions à effectuer à partir d'un opérateur, et l'apprentissage par imitation, permettant à un agent à reproduire une politique. Nous avons expliqué que l'apprentissage par renforcement permet de calculer une politique pendant la mission, en combinant optimisation des actions et test de nouvelles actions. Nous avons également présenté l'apprentissage par renforcement inverse, visant à apprendre la fonction de récompense permettant d'obtenir une politique ou un plan donné.

Nous avons étudié ces approches afin de faire évoluer le modèle de l'humain pour estimer

sa politique au cours de l'exécution. Seulement, ces approches ne sont pas idéales pour notre problème. En effet, l'apprentissage par renforcement suppose que l'agent déduise une utilité par rapport à l'état de l'environnement, et maximise sa politique à partir de celle-ci, hors il ne s'agit pas de l'utilité du modèle de l'opérateur, mais du modèle de l'agent. Il calculera donc une politique pour lui optimale. L'apprentissage par renforcement inverse, quant à lui, nécessite d'avoir la politique ou le plan complet de la mission pour calculer la fonction de récompense, ce qui est impossible au cours de l'exécution. Nous présenterons, dans le chapitre 4, des méthodes s'appuyant sur celles-ci pour mettre à jour le modèle de l'humain et pour calculer une estimation de sa politique.

Semi-autonomie

Enfin, nous nous sommes intéressés à la capacité de l'opérateur à prendre le contrôle du robot. Pour cela, nous avons défini ce qu'est la semi-autonomie, c'est à dire un niveau d'autonomie où l'agent et l'humain se partagent le contrôle du robot. Nous avons distingué la semi-autonomie à l'exécution, où l'opérateur peut intervenir pendant la mission, et la semi-autonomie à la planification, où l'opérateur influence le calcul de la politique, en envoyant des restrictions, ou des conseils à l'agent.

Nous avons ensuite expliqué ce qu'est l'autonomie ajustable, où l'opérateur peut prendre le contrôle du robot, et plus précisément l'autonomie adaptable, jugée préférable, où l'agent demande à l'opérateur d'interagir.

Nous avons présenté plusieurs méthodes pour mettre en place l'autonomie ajustable, comme l'initiative mixte, où l'agent évalue quand l'opérateur sera plus efficace que lui pour accomplir une tâche. D'autre méthodes proposent de demander l'intervention lorsque le robot perd en efficacité, ou lorsque celui-ci présente des défauts de fonctionnement. Seulement, aucune de ces méthodes ne permet à l'opérateur de définir des restrictions sur l'autonomie de l'agent.

Nous proposerons, dans le chapitre 5, des méthodes permettant à un opérateur de définir des conditions sur l'autonomie du robot.

Deuxième partie

Contribution

Introduction

Dans un système composé d'agents à autoonomie ajustable, nous nous confrontons au problème de la coordination d'agents à niveaux d'autonomie différents. Dans ce contexte, nous nous intéressons d'abord à l'ntroducton de différents niveaux d'autonomies, puis nous nous focalisons sur le problème de l'estimation de la politique des agents à autonomie zéro, ou dits télé-opérés. L'objectif est de permettre aux autres agents de se coordonner en calculant la meilleure réponse "Best-Response" à partir de cette politique estimée.

Nous parlerons dans un premier temps de la prédiction de la politique d'un agent non optimal, pour permettre à un agent autonome de s'adapter à un agent non-autonome. Nous présenterons les modèles `FORCE`, `DIST`, `LEARN` et leurs dérivées pour estimer la politique d'un agent non-optimale.

— `FORCE` permet de générer une politique en prenant en compte les dernières actions observées.
— `LEARN` essaie d'apprendre les préférences sur les états de l'opérateur à partir des actions observées.
— `DIST` est une version locale de l'apprentissage de `LEARN`.

Nous présenterons ensuite de nouveaux modèles pour permettre à un agent de planifier ses actions en fonction de restrictions, imposées par l'opérateur, comme l'interdiction d'effectuer des actions sans surveillance ou autorisation. Nous proposerons 3 modèles :

— Le premier se basera sur une modification de la fonction de transition pour gérer les interactions et le respect des contraintes.
— Le second considérera séparément le contrôle du robot et de l'autonomie.
— La dernière proposera une solution pour éviter un blocage du robot en cas de refus de l'opérateur.

Chapitre 4

Estimation d'une politique non-optimale

Sommaire

Dans un système multi-agent hétérogène, les agents autonomes doivent collaborer avec des agents non-autonomes afin d'accomplir leur objectif. Les agents autonomes, doivent être en mesure de prévoir les actions des autres pour pouvoir se coordonner avec eux.

Pour prédire les actions d'un agent autonome, il suffit de connaître, ou de supposer sa politique optimale. Seulement, lorsque l'agent est télé-opéré, la politique appliquée est celle de l'opérateur, qui peut ne pas être optimale.

Nous expliquerons dans ce chapitre en quoi la politique d'une personne diffère de celle d'un agent autonome. Ensuite, nous proposerons plusieurs méthodes visant à apprendre la politique appliquée par l'opérateur. Une est basée sur l'apprentissage des actions observées à chaque état, et deux autres basées sur les états atteignables en fonction des actions observées.

4.1 Différences entre agent autonome et télé-opéré pour la prédiction

Pour estimer la politique d'un agent autonome, il suffit de connaître son objectif, puis de modéliser et de résoudre le problème de planification. Par exemple, [Le Guillarme, 2016] propose un framework avec pour objectif de planifier les actions de défenseurs cherchant à couvrir les objectifs d'attaquants. Pour cela, dans un premier temps, les défenseurs observent les actions des attaquants pour connaître leur cible, afin d'estimer leur politique à partir d'un modèle de planification, en prenant la cible estimée comme objectif. Il leur est alors possible de s'adapter à ce comportement.

Néanmoins, une personne est différente d'un agent autonome sur divers points. En effet, l'Homme n'a pas la même appréciation des dustances que le robot, et il ses émotions, tel que le stress peuvent impacter ses décisions. Nous ferons référence, au cours de cette section, à l'agent qui estime la politique d'un autre comme d'un *observateur*. Nous distinguerons également un agent autonome en parlant d'agent, et d'un opérateur pilotant un agent en faisant référence à l'opérateur.

4.1.1 Le calcul de l'utilité

La précision de l'opérateur n'est pas la même que celle d'un ordinateur. En effet, lorsque l'agent planifie ses actions, il le fait sur un horizon fini, ou infini, de sorte à prévoir l'ensemble de la mission. Celui-ci se base sur des modèles quantitatifs, c'est à dire sa récompense et la fonction de transition, qui sont connues. L'opérateur, au contraire, peut connaître le coût ou la récompense de certaines actions, mais n'aura que très rarement une information complète sur la récompense espérée d'une politique. Il utilise souvent une approche quantitative.

Par exemple, dans le cas du déplacement du robot, un agent planifiant avec un MDP calcule la distance minimale en maximisant l'utilité, lui permettant ainsi de connaître clairement la politique optimale. L'opérateur, lui, ne calculant pas directement l'utilité du chemin parcouru, n'en fait en général qu'une estimation. Il est alors possible qu'il prenne un chemin non-optimal, dû à ce manque de précision. L'autre facteur qui peut aussi influencer la décision est l'émotion. Par exemple, le stress d'une personne peut l'amener à prendre moins de risques.

De même, le temps de réaction entre un robot et une personne n'est pas le même, cela peut influer sur la politique observée. En effet, le temps de réponse du robot dépend de la complexité temporelle de ses algorithmes, alors que pour l'opérateur, ça dépend de plus de plusieurs facteurs émotionnels.

4.1.2 Les connaissances

Dans un premier temps, les connaissances de l'agent autonomes sont apportées par les données de ses capteurs, qui sont souvent stockés et traités dans une base de connaissance

L'opérateur peut à la fois profiter des capteurs de l'agent qu'il pilote, mais aussi des informations qu'il peut avoir via la communication, ou via ses propres sens. Par exemple, l'opérateur peut s'apercevoir qu'un chemin est bloqué par des gravas. Si cette connaissance n'est pas apportée à l'observateur, alors celui-ci ne sera pas en mesure de comprendre que l'observé ne prendra pas ce chemin. Ces connaissances peuvent influer sur l'objectif de l'opérateur, ou sa "fonction de récompense", et l'inciter donc à dévier de la politique optimale.

4.1.3 L'évolution

La politique exécutée par un agent autonome étant optimale, celle-ci n'a pas lieu de changer, à moins que le modèle de l'environnement ne change (il faudra alors replanifier). Pour un opérateur, le problème est différent. En effet, il apprend, évalue en temps réel les actions qu'il effectue et observe les changements de son environnement, non planifiés à l'origine. Il peut donc adapter sa stratégie.

Cela implique pour l'observateur de mettre à jour l'estimation de la politique au cours de la mission.

4.1.4 Les émotions

Les décisions d'un agent au cours de sa mission sont prises en suivant une politique précalculée, et sont libres de toute influence émotionnelle. Par contre, l'opérateur, lui, est soumis à ces émotions, perturbant ainsi les décisions prises. Il peut par exemple hésiter, dans une situation de stress, voir même paniquer et faire une suite d'actions incohérentes.

De même, l'opérateur peut avoir des préférences personnelles, comme un chemin caché par une forêt. Si l'observateur n'a pas l'information quant à la couverture du chemin, celui-ci peut considérer ce chemin sous-optimal et donc en prédire un autre à la place.

Pour estimer la politique de l'opérateur, il devra à la fois faire face aux préférences de l'opérateur, mais aussi à ses perturbations dues aux hésitations, au stress, ... Ces perturbations sont par ailleurs uniques pour chaque opérateur, en fonction de son expérience, ou de ses préférences.

Pour estimer la politique d'un opérateur, il faut alors une méthode capable de s'adapter à l'évolution de l'opérateur, qui s'adapte à ses préférences et qui soit tolérante à ses perturbations, par exemple en situation de stress. Toutes ces raisons rendent le cas de l'estimation de la politique un cas unique présentant un challenge scientifique aujourd'hui. Nous présentons dans la partie suivante des modèles réalisés pour apprendre la politique observée afin d'estimer la politique de l'opérateur.

4.2 Nouvelles méthodes d'estimation d'une politique

4.2.1 Principe général

Nous considérons le scénario où un agent autonome observe un agent télé-opéré. Comme, l'autonomie des agents est ajustable, le robot piloté a aussi une fonction autonome. Nous considérons que l'observateur connaît le modèle du robot de son comportement pour une autonomie totale.

Ainsi, nos méthodes estimeront à l'origine que l'opérateur suit la politique optimale π_{init} calculée à partir du modèle connu. Par la suite, l'observateur perçoit, au cours d'une transition, une action observée a_o et un état atteint s_o, l'observateur mettra à jour ses connaissances sur l'opérateur. Suite à cette observation, il recalcule une politique estimée π_{est} à partir de ces connaissances acquises, si l'action observée diffère de celle prévue.

Toutes les observations, permettant d'analyser le comportement de l'opérateur afin de s'y adapter, sont enregistrées dans des historiques $H_s(t_h)$ et $H_a(t_h)$, contenant respectivement les états et les actions. La variable t_h correspond au nombre de transitions observées depuis la dernière mise à jour. On considère k la taille de l'historique.

Nous présentons ici 3 modèles pour mettre à jour la politique estimée. La première, dite `FORCE`, est une méthode de mise à jour de la politique basée sur la mémorisation des dernières

actions effectuées. Les deux autres, `LEARN` et `DIST`, ont pour objectif d'apprendre les préférences de l'opérateur sur les états du robot, en considérant une représentation factorisée d'états par des variables.

4.2.2 `FORCE`

Description

`FORCE` est une méthode mettant à jour la politique estimée en mettant à jour la politique en fixant la dernière action observée à chaque état déjà atteint au cours de l'exécution. Le calcul se fait à partir de l'équation 4.1.

$$\pi_{\texttt{FORCE}} = \begin{cases} H_a(\max_{t_h} | s_o = H_s(t_h)) & \forall s_o \in H_s \\ \operatorname{argmax}_{a \in A} r(s_t,a) + \gamma \sum_{s_{t+1} \in S} T(s_t,a,s_{t+1}) V_{\pi \pi_{\texttt{FORCE}}}(s_{t+1}) & \forall s_t \notin H_{s_t} \end{cases} \tag{4.1}$$

Ainsi, dans le cas où l'état a déjà été atteint, alors l'action correspondante dans la politique estimée est la dernière action observée à celui-ci. Dans le cas contraire, l'action correspondante est l'optimale, considérant les états où la politique est déjà définie.

Au cours de l'exécution, l'apprentissage suit l'algorithme 6. A chaque déviation de la politique observée par rapport à la politique estimée, l'agent recalcule la politique à partir d'un Value Iteration. L'algorithme prendra moins de temps qu'une planification classique, étant donné que pour chaque état visité, on ne compare plus les résultats de l'ensemble des actions possibles, mais on n'évalue que la dernière action observée à cet état pour planifier les actions des états non atteints.

Algorithme 6 : Mise à jour de la politique à partir de `FORCE`

```
1  last_action ← ∅ ; tant que objectif non atteint faire
2      s ← état courant;
3      a_e ← action estimée;
4      a_o ← action observée;
5      si a_e ≠ a_o alors
6          last_action[s] ← a_o;
           /* Application de Value Iteration                    */
7          ;
8          tant que <Conditions> faire
9              pour chaque s_t ∈ S faire
10                 si s_t ∈ last_action alors
11                     π_FORCE(s_t) ← last_action[s_t];
12                     V_{π π_FORCE}(s_t) ←
                         r(s_t,π_FORCE(s_t)) + γ Σ_{s_{t+1}∈S} T(s_t,π_FORCE(s_t),s_{t+1}) V'_{π π_FORCE}(s_{t+1});
13                 sinon
14                     π_FORCE(s_t) ← argmax_{a∈A} r(s_t,a) + γ Σ_{s_{t+1}∈S} T(s_t,a,s_{t+1}) V'_{π π_FORCE}(s_{t+1});
15                     V_{π FORCE}(s_t) ← max_{a∈A} r(s_t,a) + γ Σ_{s_{t+1}∈S} T(s_t,a,s_{t+1}) V'_{π π_FORCE}(s_{t+1});
16 retourner π_FORCE
```

La politique estimée correspond alors à la dernière politique observée à chaque état, combinée à l'adaptation du reste de la politique au comportement observé, afin d'atteindre l'objectif de manière optimale.

Limites

Cette méthode permet d'apprendre directement l'action observée aux états atteints. Néanmoins, elle ne permet pas de généraliser pour les états non atteints. Elle ne permet donc pas de prédire directement la politique de l'opérateur pour les sous-espaces de l'environnement non parcouru.

4.2.3 `LEARN`

Description

L'intuition venant à s'orienter vers cette méthode est de considérer les préférences de l'opérateur. Plutôt qu'envisager que l'opérateur choisisse une action, nous considérons qu'il choisit une destination. En, effet, l'opérateur doit choisir ses actions en fonction des états désirés qu'il veut atteindre à partir de celles-ci, ou des états indésirables qu'il pourrait atteindre en choisissant une autre action. L'objectif de cette méthode est donc d'apprendre les préférences de l'opérateur sur les états. Pour cela, nous considérons les variables composant un état, en considérant que ce sont sur celles-ci que vont se porter les préférences de l'opérateur. Par exemple, si les actions choisies par l'opérateur visent à suivre un chemin plus lumineux mais moins optimal qu'un chemin sombre, alors learn apprendra que l'opérateur préfère les états dans lesquels le robot est dans une zone lumineuse.

Soit X l'ensemble de n variables composant les états du système. Chacune de ces variables $x_i \in X$ est définie dans un domaine D_i. L'idée est d'attribuer une récompense à chaque variable qui aurait pu intéresser l'opérateur, et d'émettre un coût à chaque composante de l'état qui aurait pu déplaire à l'opérateur. Ainsi, en changeant la fonction de récompense, la politique devrait évoluer en conséquence. $Q_i(x_i(s))$ est le score de préférence de l'opérateur pour la variable x_i de l'état s. Initialement, nous considérons que l'opérateur n'a pas de préférences particulières, la table Q est donc initialisée à 0 . c est une constante, permettant de calibrer le taux d'apprentissage de la méthode de prédiction. Plus c a une valeur élevée, plus l'algorithme accordera de l'importance aux déviations de l'opérateur vis à vis de la politique estimée. Ainsi, une valeur élevée implique des changements rapides de la politique après observation.

La mise à jour des préférences de l'opérateur se déroule telle que décrite dans l'algorithme 7. Lorsque des différences sont observées entre l'action prédite et l'action observée d'une transition, alors l'algorithme met à jour la préférence de l'opérateur sur les variables de l'état. Ainsi, les valeurs des variables des états atteignables à partir de l'action observées se voient recevoir une récompense, en ligne 5, en fonction du coefficient d'apprentissage et de la probabilité d'obtenir cette valeur pour la variable donnée à partir de cette action. Inversement, en ligne 8, c'est un malus qui est appliqué vis à vis de l'action estimée.

De cette façon, cette méthode permet de considérer une préférence sur les valeurs des variables dont le système a le plus de chances de se rapprocher à partir de a_o, et un coût pour les valeurs des variables dont le système a le plus de chances de s'éloigner en déviant de la politique prédite.

A partir de ces préférences, nous définissons une nouvelle fonction d'utilité r_l, décrite en

Algorithme 7 : Mise à jour de la politique estimée

```
pour t_h ∈ [k..0] faire
    si H_a(t_h) ≠ π_est(H_s(t_h)) alors
        pour s' successeur de (H_s(t_h),H_a(t_h)) faire
            pour i ∈ [0..n] faire
                Q_i(x_i(s')) = Q_i(x_i(s')) + T(s'|H_s(t_h),H_a(t_h)) × c;
        pour s' successeur de (H_s(t_h),π_est(H_s(t_h)) faire
            pour i ∈ [0..n] faire
                Q_i(x_i(s')) = Q_i(x_i(s')) − T(s'|H_s(t_h),π_est(H_s(t_h)) × c;
si Mise à jour des préférences alors Mise à jour de la politique;
Réinitialisation historique;
```

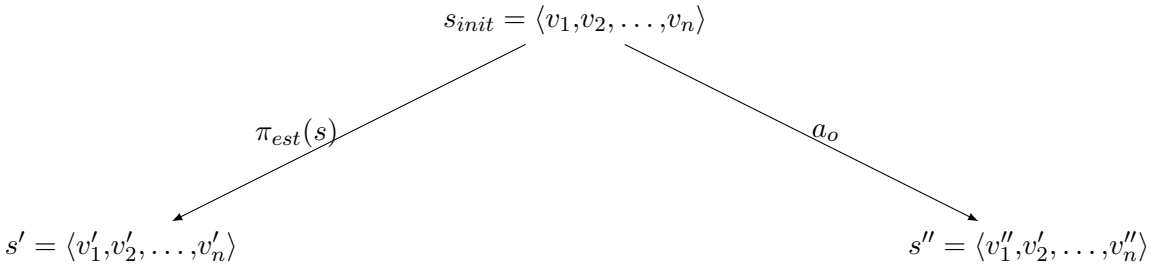


FIGURE 4.1 – Exemple formel de déviation de l'opérateur vis à vis d'une politique estimée.

équation 4.2, composé de la fonction d'utilité initiale et des préférences de l'opérateur.

$$r_l(s,a) = r(s,a) + \sum_{x_i \in X} Q_i(x_i(s)) \tag{4.2}$$

Le calcul de la nouvelle politique estimée π_{est} se fait en résolvant le MDP $\langle S,A,T,r_l \rangle$.

Exemple

Considérons, comme exemple, la figure 4.1, où l'opérateur, à un état s_{init}, ne choisit pas l'action estimée $\pi_{est}(s)$ le conduisant à un état s', mais une action a_o le menant à un état s''.

Dans cet exemple, toutes les variables composant s' et s'' sont différentes, excepté pour la seconde. En suivant cette méthode de mise à jour de la politique, les préférences de l'agent seraient modifiées telle que décrite en équation 4.3. Nous notons Q' le tableau de préférences après modifications. Comme $x_2(s') = x_2(s'')$, celle-ci reçoit à la fois le bonus et le malus de chaque transition. Sa préférence n'est donc pas changée.

$$Q_i{}'(x) = \begin{cases} Q_i(x) + c & \forall 0 \leq i < n \text{ et } i \neq 2 \text{ et } x = x_i{}'' \\ Q_i(x) - c & \forall 0 \leq i < n \text{ et } i \neq 2 \text{ et } x = x_i{}' \\ Q_i(x) & \text{sinon} \end{cases} \tag{4.3}$$

La mise à jour de l'utilité des états $s \in S$ de la fonction r_l est décrite dans l'équation 4.4.

Nous notons $r_l{}'$ la fonction d'utilité après mise à jour.

$$\begin{aligned}
&r_l{}'(s = \langle v_1'', *, \ldots, * \rangle) = r_l(s) + Q_1(v_1'') \\
&\vdots \\
&r_l{}'(s = \langle *, *, \ldots, v_n'' \rangle) = r_l(s) + Q_n(v_n'') \\
&r_l{}'(s = \langle v_1', *, \ldots, * \rangle) = r_l(s) + Q_1(v_1') \\
&\vdots \\
&r_l{}'(s = \langle *, *, \ldots, v_n' \rangle) = r_l(s) + Q_n(v_n')
\end{aligned} \tag{4.4}$$

Limites

Cette méthode offre la possibilité d'apprendre des préférences de l'opérateur sur l'ensemble de l'espace d'état, permettant ainsi de généraliser rapidement des règles de préférence. Par exemple, si l'opérateur préfère ne pas allumer les lampes du robot, l'agent devrait vite comprendre que cette action n'est pas préférée.

Néanmoins, cette généralisation est dangereuse, notamment dans le cas où l'opérateur hésite. En effet, apprendre une hésitation sur l'ensemble de l'espace d'état risque d'entraîner des parasites sur l'apprentissage de la politique à estimer. De même, le manque d'informations sur les états peut entraîner une boucle d'apprentissage.

4.2.4 `DIST`

Pour remédier au risque de parasites expliqué au paragraphe précédent, nous proposons une troisième méthode, proche de `LEARN`, mais limitant la mise à jour de récompense aux états environnant la transition où l'opérateur a dévié de la politique prédite. Pour cela, il nous faut définir la distance entre deux états.

Définition 10. *La distance $\mathcal{D}$ entre deux états correspond au nombre de variables de ces états dont la valeur est différente.*

Plus formellement, $\mathcal{D}$ est définie par l'équation 4.5. Par exemple, une distance de 0 équivaut à deux états identiques, tandis qu'une distance de n correspond à deux états sans points communs. On peut envisager de ne modifier les fonctions de récompense suite à une déviation de la politique calculée que des états d'une distance inférieure ou égale à un nombre $\delta \leq n$, par rapport à l'état où la personne aurait du se retrouver. Ainsi, nous diminuons l'influence d'une déviation sur l'ensemble du système.

$$\mathcal{D}(s,s') = \sum_{i=1}^{n} \Delta(x_i(s'),x_i(s'')),\ avec\ \Delta(a,b) = \begin{cases} 1 & si\ a \neq b \\ 0 & sinon \end{cases} \tag{4.5}$$

Pour limiter la mise à jour de la politique, il n'est plus possible de stocker toutes les préférences dans Q. En effet, comme l'application de ces préférences dépendent de la distance, celles-ci doivent concerner l'état lui-même, et pas des variables de ces états. Nous définissons alors Q la table de préférence sur les états. Soient S_{obs} et S_{est} les ensembles d'états atteignables respectivement par l'action observée ou estimée. La mise à jour de Q, pour un état s sera soumise à condition :

— s doit être dans S_{obs} ou S_{est}.
— Si $s \in S_{obs}$ (respectivement S_{est}), alors $Q_s(s)$ est mis à jour avec le bonus (respectivement le malus) associé à l'action observé (respectivement estimé), ainsi que le malus (respectivement bonus) de l'autre action s'il est assez proche des états S_{est} (respectivement S_{obs}).

Pour respecter la seconde partie de la dernière condition, nous utilisons la fonction $\mathcal{D}_{ok}$, définie dans l'équation 4.6, vérifiant qu'un état est proche d'un ensemble d'état.

$$\mathcal{D}_{ok}(s,S) = \begin{cases} 1 & si \ \min\limits_{s' \in S} \mathcal{D}(s,s') < \delta \\ 0 & sinon \end{cases} \tag{4.6}$$

Etant donné une déviation, la mise à jour de la politique se fait alors en deux étapes. La première consiste à mettre à jour la fonction d'utilité pour chaque transition, en suivant l'équation 4.7. La seconde consiste à appliquer l'utilité associée aux états proches de l'ensemble opposé, comme décrit en équation 4.8.

$$r'(s) = r(s) + \sum_{i|x_i(s)=x_i(s)} Q_i(x_i(s)) \forall s \in S_o \cup S_p \tag{4.7}$$

$$\begin{aligned} r'(s_o) &= r(s_o) + \mathcal{D}_{ok}(s_o,S_{est}) \sum_{i|\exists s_p \in S_{est}|x_i(s_o)=x_i(s_p)} Q_i(x_i(s_o)) \forall s_o \in S_{obs} \\ r'(s_p) &= r(s_p) + \mathcal{D}_{ok}(s_p,S_{obs}) \sum_{i|\exists s_o \in S_{obs}|x_i(s_o)=x_i(s_p)} Q_i(x_i(s_p)) \forall s_p \in S_{est} \end{aligned} \tag{4.8}$$

Discussion

La méthode `DIST` permet de diminuer la propagation des récompense, évitant une généralisation trop importante de l'apprentissage. En effet, plutôt qu'apprendre de chaque action dans l'ensemble des états possible, cet apprentissage ne se fait que pour les états similaires à la situation en cours, étant donné qu'il faut un certains nombres de variables du systèmes identiques.

Néanmoins, la propagation est beaucoup plus faible et finit par se limiter aux états de la transition. Cette limite assure une limitation du sur-apprentissage de la politique, mais comme seuls les états proches et atteignables depuis l'état courant bénéficient de l'apprentissage, l'estimation de la politique n'est améliorée que localement.

Au cours des expérimentations, bien que les méthodes `FORCE`, `LEARN` et `DIST` se soient montrées efficaces, celles-ci ont démontrées que les deux dernières méthodes sont plus instables voir peu efficaces. C'est pourquoi, dans la section suivante, nous détaillons les sources des instabilités, et nos approches pour améliorer nos algorithmes.

4.3 Amélioration des algorithmes

Les méthodes `LEARN` et `DIST` permettent d'apprendre des préférences de l'opérateur sur les états. Seulement, au cours des différentes expériences sur l'estimation à partir de cet algorithme, nous avons pu observer plusieurs comportements indésirables avec `LEARN` et `DIST` :

— L'opérateur peut faire des choix qui ne dépendent pas des variables composant l'espace d'états du système. Par exemple, le robot peut ne pas percevoir l'éclairage environnant, bien que celui-ci peut être un critère déterminant pour l'opérateur, qui profite d'un flux

vidéo extrait d'une caméra embarqué. Cet éclairage n'étant pas perçu par le robot, celui-ci n'est pas inclus dans les variables composant l'espace d'états du système. Dans ce genre de situations, l'observateur apprenant avec `LEARN` des préférences sur les variables du système peut rester bloqué sur du sur-apprentissage, à cause de la généralisation de celui-ci.

— Au cours de l'apprentissage, à force d'augmenter l'estimation de la récompense estimée de l'opérateur associée à une compétence du système, l'utilité de certains états devenaient plus importante dans le modèle, entraînant la politique générée par `LEARN` ou `DIST` à estimer que l'opérateur abandonne l'objectif pour se diriger vers ces états.

— `DIST`, supposé plus stable que `LEARN`, s'est montré plus instable que prévu au cours des expérimentations. Nous expliquons dans 4.3.1 les causes de cette instabilité et les améliorations apportées à cette méthode pour stabiliser l'estimation de la politique de l'opérateur.

Nous détaillons ci dessous les problèmes de stabilité rencontrés ci-dessus, et présentons trois améliorations que nous avons apportées à ces méthodes.

4.3.1 Nouvelle version de `DIST`

Au cours de nos expériences, nous avons pu observer une certaine instabilité de `DIST`. Pour identifier la cause de l'instabilité, prenons un exemple.

Exemple

y\x	1	2
1	s_3	s_4
2	s_1	s_2

(a) Espace d'état

state\action	Up	Right
s_1	0	0
s_2	0	1
s_3	0.9	0
s_4	0.1	0

(b) Fonction de transition depuis s_1

Etat	Utilité
s_1	0
s_2	1
s_3	0
s_4	0

(c) Fonction d'utilité

Tableau 4.1 – Exemple de MDP

Prenons pour exemple le MDP en tableau 4.1 composé de :

— Espace d'état $S = \{s_1,s_2,s_3,s_4\}$. Les variables des états sont décrits dans le tableau 4.1a.

— Ensemble d'actions $\{Up,Right\}$.

— Fonction de transition T, décrite en table 4.1b.

— Fonction d'utilité r, décrit en table 4.1c.

Nous considérons le scénario suivant. Le robot est piloté à l'état s_1. Première action de l'opérateur, l'observateur estime qu'il suivra la politique initiale (donc optimale), et devrait effectuer Right. Hors, il le voit effectuer Up. En utilisant la méthode `DIST`, l'observateur en déduit une préférence sur les variables des états décrite dans le tableau 4.2.

En appliquant `DIST` au cours de cette dérivation, nous obtenons la fonction d'utilité suivante :

— $Q'(s_1) = Q(s_1)$

— $Q'(s_2) = Q(s_2) + Q_x(2) + Q_y(2) + \underbrace{Q_x(2)}_{\text{proche de } s_4}$

— $Q'(s_3) = Q(s_3) + Q_x(1) + Q_y(1)$

		Q^+	Q^-
x	1	$0.9 \times c$	0
	2	$0.1 \times c$	$-c$
y	1	c	0
	2	0	$-c$

Tableau 4.2 – Préférences des variables sur les états.

— $Q'(s_4) = Q(s_3) + Q_x(2) + Q_y(1) + \underbrace{Q_x(2)}_{\text{proche de } s_2}$

Nous pouvons observer que, plutôt que réduire l'écart d'utilité entre les états proches, ceux-ci reçoivent une récompense fixe, entraînant un excès d'apprentissage.

Amélioration

Pour éviter ce problème, il est nécessaire de distinguer les préférences apprises de l'action observée (les bonus) de celles apprises de l'action estimée (les malus), afin de les appliquer séparément en fonction des distances. Nous définissons pour cela Q^+ la table de préférence sur les variables d'états associée aux bonus et Q^- celle associée aux malus. Elles sont calculées à partir des équations 4.9 et 4.10, étant donné une transition à partir d'un état s_{init} et une action observée a_o.

$$\forall i \in [1..n], \forall v \in D_i : Q_i^+(v) = c \times \sum_{s_o \in S_{obs} | x_i(s_o) = v} T(s_{init}, a_o, s_o) \tag{4.9}$$

$$\forall i \in [1..n], \forall v \in D_i : Q_i^-(v) = -c \times \sum_{s_e \in S_{est} | x_i(s_e) = v} T(s_{init}, a_o, s_e) \tag{4.10}$$

A partir de ces tables de préférences sur les variables, nous calculons une nouvelle table de préférence Q sur les états du système grâce aux équations 4.11 et 4.12, où Q' est la table Q après mise à jour de la table.

$$\forall s_o \in S_{obs} : Q'(s_o) = Q(s_o) + \sum_{i=1}^{n} \left[Q_i^+(x_i(s_o)) + \mathcal{D}_{ok}(s_o, S_{est}) \times Q_i^-(x_i(s_o)) \right] \tag{4.11}$$

$$\forall s_e \in S_{est} : \forall Q'(s_e) = Q(s_e) + \sum_{i=1}^{n} \left[Q_i^-(x_i(s_e)) + \mathcal{D}_{ok}(s_e, S_{obs}) \times Q_i^+(x_i(s_e)) \right] \tag{4.12}$$

La table Q nous permet de définir une nouvelle fonction d'utilité $r_d(s,a) = r(s,a) + Q(s)$, utilisée pour calculer une nouvelle estimation pour la politique de l'opérateur.

Etat	$\delta = 0$	$\delta = 1$
s_1	0	0
s_2	$-2 \times c$	$-1.9 \times c$
s_3	$1.9 \times c$	$1.9 \times c$
s_4	c	$0.1 \times c$

Tableau 4.3 – Préférences sur états en fonction de δ.

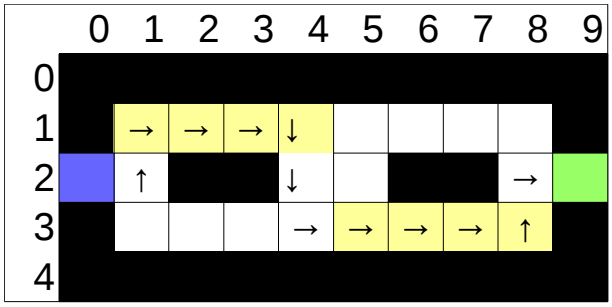


FIGURE 4.2 – Exemple d'instabilité du à un manque d'informations

En prenant en compte ces modifications, nous obtenons les utilités décrites en table 4.3. Pour $\delta = 1$, ces valeurs sont calculées de la façon suivante :

— $Q'(s_1) = Q(s_1)$
— $Q'(s_2) = Q(s_2) + Q_x^-(2) + Q_y^-(2) + \underbrace{Q_x^-(2)}_{\text{proche de } s_4} = Q(s_2) - 1.9 \times c$
— $Q'(s_3) = Q(s_3) + Q_x^+(1) + Q_y^+(1) = Q(s_3) + 1.9 \times c$
— $Q'(s_4) = Q(s_3) + Q_x^+(2) + Q_y^+(1) + \underbrace{Q_x^-(2)}_{\text{proche de } s_2} = Qs_4 + 0.1 \times c$

Ainsi les méthodes voient l'écart d'utilité entre les états proches plus ou moins diminués, en fonction de δ, et de la fonction de transition. Cette modification permet d'améliorer la stabilité de l'algorithme en équilibrant les modifications de la fonction de récompense en fonction de la fonction de transition. Ainsi, l'impact de la déviation pour l'interprétation des préférences de l'opérateur sur un état est plus important quand le robot a de fortes chances d'arriver dans cet état, suite à l'action estimée ou observée.

4.3.2 Ajout d'informations par l'opérateur

Apprendre sur les variables des états est contraignant, car elles reflètent l'interprétation du robot de son environnement, alors que l'opérateur a sa propre interprétation, vu qu'il le perçoit différemment. L'opérateur a alors ses propres critères de décisions que l'observateur ne possède pas, puisqu'elles ne font pas parti du modèle de décision de l'agent autonome. Par exemple, la figure 4.2 montre un cas où l'observateur ne peut apprendre la politique de l'opérateur sans informations supplémentaires. Considérons les cases noires des obstacles, la case bleue la case de départ, la case verte l'état objectif et les cases jaunes des cases mieux éclairées. Si l'observateur n'a pas les informations d'éclairage, il apprend en premier lieu à préférer passer par le nord. Au milieu du parcours, l'opérateur change de direction, réinitialisant la préférence au nord et augmentant l'estimation de la préférence de l'opérateur envers le sud. Si l'opérateur doit refaire la mission, les mêmes erreurs de répétitions vont être répétées. L'observateur n'arrivera donc pas à apprendre la politique de l'opérateur.

Pour éviter ce problème, nous proposons de permettre à l'opérateur d'enrichir les connaissances de l'agent sur les états du système en ajoutant des informations associées aux états.

Pour cela, nous associons aux états un ensemble de **paramètres** P. Ainsi, un état est décrit par n_p paramètres p_i, définis dans un domaine D_i^p. $p_i(s)$ est la valeur du paramètre i dans l'état s.

Ces paramètres n'étant pas connus initialement par l'agent, c'est à l'opérateur de les définir en fonction de ses besoins. Il peut par exemple être utilisé pour définir le paramètre "éclai-

rage", décrivant l'éclairage environnant du robot. Ce paramètre peut prendre plusieurs valeurs, correspondant au niveau d'éclairage (sombre, éclairage faible, lumineux ...). Ces informations correspondent aux critères de l'opérateur, et permettent à l'observateur d'estimer une nouvelle politique en prenant en compte les préférences de l'opérateur vis à vis de ces paramètres. Pour cela, il est nécessaire d'associer une fonction d'utilité à ces paramètres.

La structure des paramètres étant la même que celles des variables composant l'espace d'état, les formules et algorithmes, définis pour `LEARN` et `DIST`, en section 4.2.3, 4.2.4 et 4.3.1, afin de mettre à jour l'utilité estimée de l'opérateur et la politique associée peuvent également être utilisées avec les paramètres. Il est ainsi possible d'estimer une politique à partir des préférences Q sur les variables composant l'espace d'états, des préférences sur les paramètres Q^p, ou une combinaisons des deux, à partir, respectivement, des équations 4.13, 4.14 et 4.15.

$$r_l(s,a) = r(s,a) + \sum_{x_i \in X} Q_i(x_i(s)) \tag{4.13}$$

$$r_l(s,a) = r(s,a) + \sum_{p_i \in P} {Q_i}^p(p_i(s)) \tag{4.14}$$

$$r_l(s,a) = r(s,a) + \sum_{x_i \in X} Q_i(x_i(s)) + \sum_{p_i \in P} {Q_i}^p(p_i(s)) \tag{4.15}$$

Dans l'exemple en figure 4.2, en prenant en compte l'éclairage, l'agent apprend à préférer la trajectoire éclairée, assurant ainsi un apprentissage rapide et avec moins d'erreur d'estimation.

4.3.3 Suppression des maximums locaux

En touchant directement à l'utilité des états, les fonctions `DIST` et `LEARN` peuvent créer des incohérences dans celles-ci. Dans le schéma en figure 4.3, l'agent part de la case bleue et doit arriver à la case verte. En suivant la trajectoire présentée dans le schéma, il dévie de la politique optimale qui consiste à choisir le chemin le plus direct. Considérons un coût de 0.1 à chaque action, un facteur d'apprentissage à 0.5 et une récompense pour atteindre l'objectif à 1. Dans la version actuelle de l'apprentissage, l'observateur va apprendre que l'opérateur préfère éviter l'état où l'agent est à gauche de l'objectif, et qu'il préfère se diriger vers le nord. Par la suite, comme le chemin alternatif pour rejoindre l'objectif est long, il va tenter de revenir en arrière, car ce sera toujours le chemin optimal. Arrivé aux coordonnées (1,0), l'observateur a appris que l'opérateur a une récompense pour rester dans la route au nord équivalente à celle d'atteindre l'objectif. Ainsi, en calculant une politique optimale associée aux préférences de l'opérateur, l'observateur va estimer que l'agent va rester sur cette route. Ainsi, la route finit par remplacer l'objectif de l'opérateur dans le modèle de l'observateur.

Pour remédier à ce problème, nous proposons une légère modification dans l'application des préférences de l'opérateur dans la nouvelle fonction d'utilité. Si nous n'appliquons que les préférences négatives à la fonction d'utilité, comme décrit dans l'équation 4.16, alors les récompenses ne devraient plus pouvoir dépasser celle de l'objectif.

$$r_p(s) = \begin{cases} 0 & if \sum_{i=1}^{n} Q_i(x_i(s)) >= 0 \\ \sum_{i=1}^{n} Q_i(x_i(s)) & otherwise \end{cases}$$

$$V(s_t) = \max_{a \in A} \sum_{s_{t+1} \in S} T(s_t,a,s_{t+1}) \times (r(s_t,a) + r_p(s_t) + V(s_{t+1})) \tag{4.16}$$

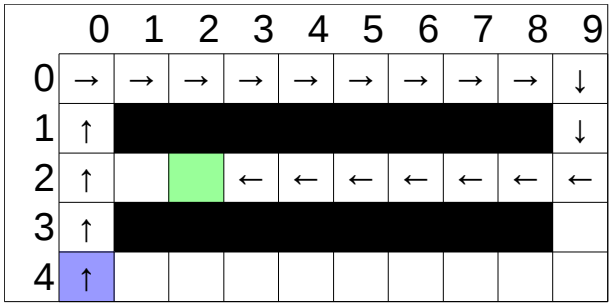


FIGURE 4.3 – Exemple de maximum local

Ainsi, les algorithmes pour mettre à jour les préférences de l'opérateur ne changent pas, et la mise à jour se fait avec n'importe quelle méthode de résolution de MDP, tel que Value Iteration (section 2.2.3).

4.4 Conclusion

Au cours de ce chapitre, nous avons présenté les problèmes suscités par la prédiction d'un agent télé-opéré par un opérateur, comme les préférences de l'opérateur, ou ses hésitations.

Pour faire face à ces problématiques, nous avons introduits trois méthodes pour mettre à jour la prédiction de la politique, à partir de l'observation du robot. `FORCE` est une méthode qui intègre les couples états/actions observés et les fixe dans la politique pour en générer une nouvelle. `LEARN` est une approche visant à apprendre les préférences de l'opérateur sur les variables des états, pour pouvoir généraliser les connaissances obtenues sur l'ensemble de l'espace d'état. `DIST` est une méthode proche de `LEARN`, mais qui limite la mise à jour des préférences de l'opérateur aux états concernés par la transition courante. Ces prédictions permettront à un observateur de s'adapter à un robot télé-opéré à l'aide d'un modèle tel que leader-suiveur.

Par la suite, pour apporter plus de stabilité dans l'apprentissage, nous avons proposé deux solutions. L'une implique d'inclure des informations de l'opérateur dans le processus de planification, et l'autre vise à limiter l'augmentation de l'utilité des états. Pour compléter ce travail, nous devons mettre en place des algorithmes de résolution de jeux stochastiques (type "Best-Response) pour caluler les politiques coordonnées des agents autonomes avec celles estimées des agents télé-opérées.

Dans la partie expérimentation, nous présenterons une analyse de ces méthodes pour analyser leur efficacité et leur stabilité, et les comparerons avec des approches de la littérature légèrement adaptées pour essayer de résoudre ce problème.

Dans ce chapitre, nous avons étudié comment prédire les actions d'un agent suivant une politique non-optimale. Dans le chapitre suivant, nous proposerons un modèle permettant de générer un comportement non-optimal, en imposant des restrictions liée à l'autonomie d'un agent à autonomie ajustable.

Chapitre 5

MDP pour l'autonomie ajustable

Sommaire

Dans ce chapitre, nous proposons un modèle étendu de MDP permettant à un opérateur de définir des restrictions sur l'autonomie d'un agent. Ainsi, certaines actions nécessiteraient un niveau d'attention de l'opérateur, ou son autorisation pour être effectuées. Par exemple, un robot muni d'une arme devrait demander l'autorisation avant de faire feu. Dans les cas extrêmes, l'agent devra être piloté.

Cet agent doit être capable, au cours de l'exécution, de demander à l'opérateur d'être **surveillé**, **piloté**, ou **autorisé à agir.** Ce modèle doit permettre à l'agent de planifier ses actions de sorte à modifier son niveau d'autonomie pour s'adapter à ces restrictions, que ce soit en demandant l'intervention d'un opérateur ou en trouvant une alternative suffisamment efficace pour pouvoir se passer de l'opérateur, en accord avec les restrictions définies.

Dans ce chapitre, nous présentons en premier lieu le problème, et présentons le comportement souhaité. Ensuite, nous formalisons le problème. Enfin nous présenterons trois modèles. Le premier se basera sur une intégration des interactions avec l'opérateur dans la fonction de transition. La seconde distinguera les variables d'états et actions associés à l'autonomie du robot de

celles liées à son contrôle. La troisième méthode présentera une modification de Value Iteration afin d'éviter le blocage du robot si l'opérateur s'obstine à refuser au robot une action.

5.1 Définition du problème

Considérons un scénario où un agent, en cours de mission, bénéficie du soutien d'opérateurs. Cet agent est en mesure d'agir de manière autonome, mais peut également demander le soutien aux opérateurs. Néanmoins, un comportement intégralement autonome n'est pas idéal, car certaines zones de son environnement sont dangereuses, et il serait risqué de le laisser évoluer de lui même. Dans l'idéal, l'agent doit demander l'autorisation, ou prévenir qu'il s'apprête à effectuer une action jugée risquée, pour que l'opérateur puisse agir en conséquence, que ce soit en surveillant le robot ou en le télé-opérant.

Certaines méthodes permettent de juger des situations où l'intervention d'un opérateur est nécessaire. Par exemple, [Mouaddib *et al.*, 2010a] propose par exemple de modéliser la fonction de transition de l'opérateur, permettant ainsi de déterminer des situations où l'opérateur est plus efficace que l'agent. [Kennedy, 1999] propose quant à lui de demander à l'opérateur d'intervenir lorsqu'il détecte un défaut au niveau des capteurs. Néanmoins, ces méthodes ont un usage restreint. Il est par exemple difficile d'obtenir une fonction de transition associée à un opérateur, à moins d'avoir à disposition des traces d'utilisation du robot sur de très longues périodes. De plus, certaines décisions peuvent nécessiter une confirmation avant exécution, comme l'usage d'une arme. De même, se baser sur les défauts de capteurs ne permet pas de prendre en compte les risques, mais de s'adapter une fois que le robot est endommagé.

C'est pourquoi nous proposons une approche permettant à un opérateur de définir les cas où l'agent doit demander son intervention. Pour cela, l'opérateur impose à l'agent des restrictions liées à son autonomie. Par exemple, l'agent doit demander l'autorisation avant d'effectuer une action, ou doit être surveillé pour entrer dans une zone. L'agent doit alors être en mesure de planifier ses actions de sorte de demander à l'avance une autorisation avant d'effectuer l'action, ou alors de trouver une alternative à cette action lorsque celle-ci est optimale. Selon le modèle SAE-I en figure 3.2 décrivant des niveaux d'autonomies pour des véhicules à pilotage automatique, ce système serait catégorisé de niveau 3, c'est à dire autonome de manière conditionnelle. Il serait autonome dans certaines situations avec peu de risques, mais demanderait l'intervention de l'opérateur dès que nécessaire.

Dans le cadre de ce problème, nous envisageons plusieurs niveaux d'autonomie pour l'agent, faisant référence à l'attention de l'opérateur. Pour cela, nous avons étudié comment positionner l'agent dans le modèle PACT, permettant de représenter plus facilement des changements d'autonomies. A partir de ce modèle, nous avons défini les 4 niveaux d'attentions de l'opérateur ci-dessous, reportés sur le modèle PACT en figure 5.1.

1. Lorsque l'opérateur est indisponible, l'agent est totalement autonome, correspondant ainsi au niveau 5 de PACT.
2. Lorsque l'opérateur est disponible, il est en attente de requêtes de l'agent. Néanmoins, celui-ci peut toujours reprendre le contrôle du robot. C'est pourquoi nous associons ce niveau d'attention à la fois au niveau 3 et 4 de PACT.
3. Lorsque l'opérateur surveille l'agent, celui-ci agit jusqu'à ce que la surveillance ne soit plus nécessaire, ou que l'opérateur l'interrompe. Nous associons ce niveau au 4 de PACT.
4. Lorsque l'opérateur télé-opère le robot, l'agent n'a plus de contrôle sur celui-ci, ce qui correspond au niveau 0 de PACT.

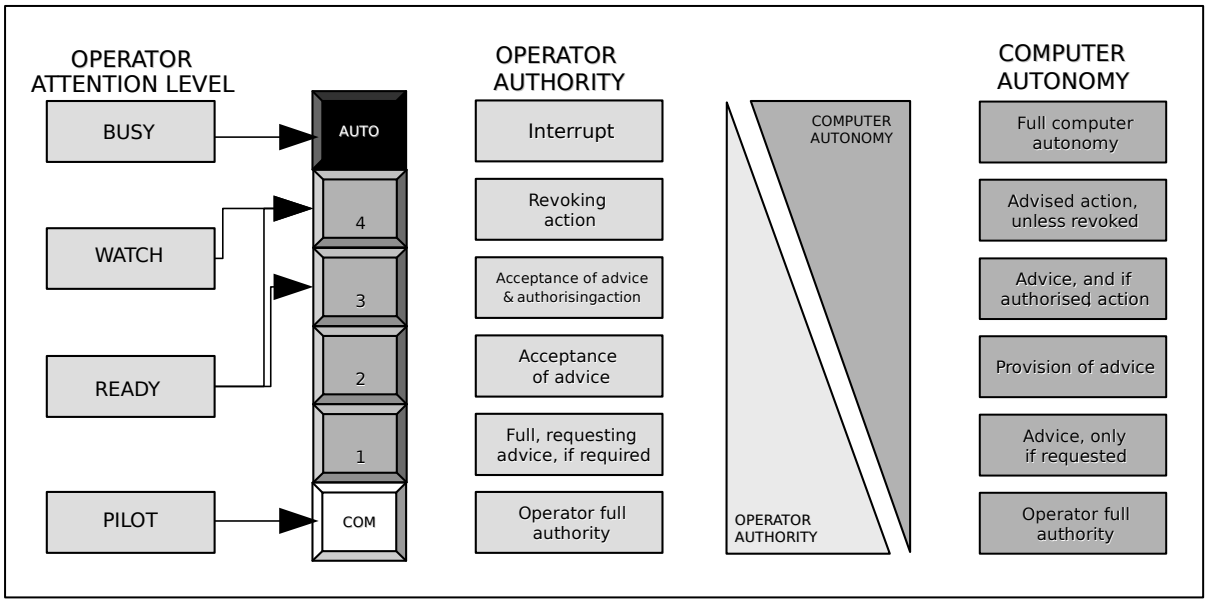


FIGURE 5.1 – Niveaux d'attention de l'opérateur porté sur le modèle PACT.

5.2 Comportement souhaité

Nous définissons dans cette section le comportement de l'agent dans des cas particuliers. Il doit être en mesure de communiquer avec l'opérateur pour demander une autorisation, ou un changement d'autonomie. Certains critères en autonomie ajustable sont conseillés par [Zilberstein, 2015], comme une transition douce d'autonomie. C'est pourquoi nous souhaitons que cette demande ne soit pas faite au dernier moment, notamment pour laisser à l'opérateur le temps d'analyser la situation avant de décider. Nous présentons, dans la suite, les différents modes de restrictions sur l'autonomie que nous envisageons.

5.2.1 Mode 1 : la demande d'autorisation

Dans certaines situations, l'agent n'a pas réellement besoin de l'aide de l'opérateur, mais doit tout de même obtenir l'autorisation avant de faire certaines actions, pour s'assurer par exemple qu'aucun imprévu ne viendrait interrompre une action importante. Lorsque l'agent demande une autorisation, 3 cas sont possibles :

— Refus : dans ce cas l'autorisation ne doit plus être redemandée.
— Accord : dans ce cas la liste des demandes rejetées est réinitialisée.
— Absence de réponse : dans ce cas, nous considérons l'opérateur dans l'incapacité de répondre à toute requête.

5.2.2 Mode 2 : changement du niveau d'autonomie

Un opérateur peut estimer dans certaines situations que laisser le robot agir seul, sans surveillance, peut être dangereux pour la mission. Il peut alors décider que dans certains états, il est nécessaire que l'agent soit surveillé avant d'agir, ou même télé-opéré. Nous souhaitons un passage progressif d'un niveau d'autonomie à un autre. Par exemple, l'opérateur n'est pas supposé prendre le contrôle de l'agent sans avoir eu le temps d'analyser la situation au préalable, et doit donc surveiller en premier lieu le robot.

Lors d'une demande de changement d'autonomie, deux cas sont possibles :

— L'opérateur est disponible, et change le niveau d'autonomie du robot, le plan de ce dernier n'est pas changé.

— L'opérateur est occupé avec un autre agent, et ne peut répondre à la requête du premier. Dans ce cas, l'agent doit avoir calculé un autre plan à l'avance pour répondre à ce problème. Si l'agent se trouve dans un état où il est restreint, alors il doit être en mesure d'effectuer une politique par défaut pour recouvrir la situation.

Pour appliquer ce changement de niveau d'autonomie, il est nécessaire dans un premier temps d'avoir certaines connaissances sur l'ensemble du système, incluant l'opérateur, et de développer un modèle permettant de respecter les règles décrites ci-dessus.

5.3 Définitions et formalisation du problème

Nous cherchons à représenter les connaissances nécessaires à l'agent pour pour lui permettre de planifier des actions de sorte à respecter des niveaux d'autonomie définis par l'opérateur. Pour cela, nous partons d'un MDP, permettant de modéliser l'agent, ses actions et son environnement, puis nous ajouterons les informations modélisant les restrictions d'autonomie définies par l'opérateur. Partir d'un MDP nous permettra de modéliser les différentes incertitudes associées au problème, comme les réponses de l'opérateur à chaque requête des agents. Il nous faudra également modéliser le niveau d'attention que porte l'opérateur sur le robot, pour savoir le niveau d'autonomie de l'agent, et d'autres connaissances de l'opérateur, pour estimer les réactions possibles de l'opérateur. Par exemple, un opérateur occupé a peu de chances de répondre favorable à une demande de télé-opération.

5.3.1 Restrictions

Dans un premier temps, il nous faut représenter les restrictions définies par l'opérateur. Ces restrictions sont des règles qui imposent à l'agent un certain niveau d'autonomie. Par exemple, un opérateur peut vouloir qu'un agent ne s'aventure pas dans une zone accidentée sans surveillance ou de manière autonome, afin de s'assurer qu'il ne se bloque pas tout seul. L'opérateur pourrait en effet l'assister en observant le flux vidéo d'une caméra. Elles peuvent être définie pour une situation, comme une zone accidentée, ou une action, comme l'usage d'une arme.

Plus formellement, une restriction $\mathfrak{R}$ est un tuple $\langle S_{\mathfrak{R}}, A_{\mathfrak{R}}, RL_{\mathfrak{R}}, \pi_{\mathfrak{R}} \rangle$, où :

— $S_{\mathfrak{R}}$ est l'ensemble des états inclus dans la restriction.

— $A_{\mathfrak{R}}$ est l'ensemble des actions inclus dans la restriction.

— $RL_{\mathfrak{R}}$ est le niveau RLde cette restriction.

— $\pi_{\mathfrak{R}}$ est la politique à suivre en cas de refus, lorsque l'agent arrive dans un cas restreint, défini $\forall s \in S_{\mathfrak{R}}$. Elle sert à remplacer l'action par défaut.

Si $A_{\mathfrak{R}} = \emptyset$, alors la restriction s'applique à toute action $a \in A$ dans les états désignés (Par exemple une zone dangereuse), et inversement si $SR = \emptyset$ (Par exemple piloter pour tirer sur une cible).

Niveau de restriction Une restriction est identifiée à plusieurs niveaux d'autonomies. En effet, toutes les situations ne nécessitent pas le même niveau d'intervention de l'opérateur. Nous avons identifié les 4 niveaux de restriction suivants :

1. `FREE` Aucune restriction
2. `REQUEST_REQUIRED` Autorisation requise.

3. `WATCH_REQUIRED` L'opérateur doit surveiller.
4. `PILOT_REQUIRED` L'opérateur doit piloter l'agent.

Outils d'exploration de la liste des restrictions

L'agent a ainsi une liste de restrictions à respecter pour la planification. Pour simplifier l'écriture de nos algorithmes à venir par la suite, nous définissons les fonctions suivantes :

— $\mathcal{R}(s,a) = \mathfrak{R} | s \in S_{\mathfrak{R}}, a \in A_{\mathfrak{R}}, \nexists \mathfrak{R}' | RL_{\mathfrak{R}'} > RL_{\mathfrak{R}}$
Retourne la restriction de plus haut niveau associée au couple $\langle s,a \rangle$, si elle existe.

— $\mathcal{R}_l(s,a) = RL_{\mathfrak{R}} | \mathcal{R}(s,a) = \mathfrak{R}$
Retourne le niveau de restriction minimum à respecter. Ainsi, l'agent ne peut pas effectuer l'action a à l'état s si son niveau d'autonomie est trop élevé, c'est à dire s'il est trop indépendant.

— $n(\mathfrak{R}) = \begin{cases} i \in [1..n_a] & si\ RL_{\mathfrak{R}} = \texttt{REQUEST\_REQUIRED} \\ 0 & sinon \end{cases}$
Retourne le numéro indexant une requête de type `REQUEST_REQUIRED`, sur les n_a requêtes de ce type. Ce numéro permet d'identifier la demande à adresser à l'opérateur, et à savoir quelle est la dernière autorisation que ce dernier lui a accordé.

— $\mathfrak{R}_n(i)$ est la restriction telle que $\mathfrak{R}_n(n(\mathfrak{R})) = \mathfrak{R}$, ou la restriction de type `REQUEST_REQUIRED` indexée au numéro i. Elle permet d'associer une requête à une restriction. Lorsque l'opérateur a reçu une autorisation, celui-ci reçoit le numéro de cette autorisation. Cette fonction lui permet d'obtenir la restriction correspondante, lui permettant de connaître les états et actions temporairement autorisés.

5.3.2 Niveau d'attention

Afin de savoir quand solliciter l'opérateur, pour respecter les restrictions, il faut connaître son niveau d'autonomie actuel. De plus, nous cherchons à éviter de solliciter un opérateur déjà occupé, faisant perdre du temps à l'agent qui attend une réponse, et perturbant l'opérateur avec de nombreuses notifications. Pour considérer ces deux points, nous représentons le niveau d'attention (`Att`) de l'opérateur pour l'agent courant. Nous identifions 4 niveaux d'attention possibles, que nous reportons sur le modèle PACT, en figure 3.1 :

1. `BUSY` L'opérateur n'est pas en mesure de répondre aux requêtes de l'agent. (PACT 5)
2. `READY` L'opérateur est en attente de requêtes des agents. (PACT 3-4)
3. `WATCH` L'opérateur surveille l'agent courant. (PACT 4)
4. `PILOT` L'opérateur pilote l'agent courant. (PACT 0)

L'opérateur sera indisponible pour les autres agents s'il est à PILOT envers un agent. Nous considérons ces niveaux d'attention ordonnés, ainsi `BUSY` < `READY`. On les considère également valués, pour simplifier les notations. Ainsi `READY` = `BUSY` +1.

5.3.3 Connaissances de l'agent sur l'opérateur

Au cours de l'exécution, afin de changer de niveau d'autonomie, l'agent doit interagir avec l'opérateur. Pour mesurer l'impact de ces interactions,faut modéliser dans la fonction de transition du MDP les réponses possibles de l'opérateur à chaque requête. Ainsi, pour planifier, on souhaite par exemple connaître la probabilité que l'opérateur accepte de surveiller l'agent s'il est près d'une zone identifiée à risque.

On considère pour l'agent les connaissances de l'opérateur suivantes :

— $P_{OK}(\mathfrak{R},\texttt{Att})$: probabilité pour l'opérateur d'autoriser l'agent à effectuer une action de la restriction dans un état de la restriction. Vaut 1 pour $RL_{\mathfrak{R}} \neq \texttt{REQUEST\_REQUIRED}$.
— $P_w(s,a,\texttt{Att})$: probabilité pour l'opérateur d'accepter la surveillance d'une action. En fonction de cette probabilité, l'agent peut décider qu'une solution alternative peut être meilleure que d'attendre l'aide de l'opérateur.
— $P_{\texttt{TO}}(s,\texttt{Att})$: probabilité que l'opérateur accepte de télé-opérer l'agent.
— $\pi_{\texttt{TO}}(s,a)$: probabilité que l'opérateur, durant la télé-opération, choisisse l'action a à l'état s. Il s'agit de la politique estimée de l'opérateur lorsqu'il prend le contrôle du robot. Ces estimations peuvent être issues des modèles d'estimations présentés en section 4.
— $c(RL)$: coût d'une requête de niveau RL. Ce coût modélise la gêne occasionnée à l'opérateur en lui demandant de l'aide. Ainsi, si l'apport de l'aide de l'opérateur est moindre, et que des solutions alternatives existent, l'agent peut estimer plus optimal une solution alternative. En général, plus l'attention requise de l'opérateur est élevée, plus ce coût est important.

Ces probabilités peuvent être définies soit par l'opérateur, soit avec des algorithmes d'apprentissage au cours de l'exécution.

5.4 Premier modèle : Restricted-MDP

5.4.1 Présentation de l'idée

Afin d'assurer le respect des restrictions au cours de l'exécution, et leur prise en compte dans la planification, nous avons choisi d'inclure les demandes de changement d'autonomie et d'autorisations directement dans la fonction de transition. La seule exception est la télé-opération, car l'agent abandonne alors le contrôle du robot.

Dans le cas où l'agent envoie une requête à l'opérateur, deux situations sont possibles :

1. L'opérateur répond favorablement à la requête. Dans ce cas, le niveau d'attention de l'opérateur atteint celui demandé par l'agent en cas de changement d'autonomie, et l'agent peut effectuer l'action qu'il entreprenait.
2. L'opérateur ne répond pas, ou refuse. Dans le cas d'une demande de changement d'autonomie, l'opérateur est alors considéré occupé. Cela permet, en plus de modéliser le comportement de l'opérateur, de s'assurer que l'agent ne répétera pas les requêtes sans arrêt.

   Pour toute requête refusée, l'agent ne peut plus faire l'action qu'il souhaitait faire. Il fera alors une action par défaut, comme par exemple rester sur place, ou sortir d'un état restreint. Cette action par défaut sert surtout à éviter que le robot ne puisse plus agir en entrant dans une zone restreinte.

Nous avons choisi cette méthode car elle permet de simplifier la planification des actions de requêtes. En effet, les actions d'interactions étant dans la fonction de transition, il n'est pas nécessaire de s'assurer que celle-ci a été effectuée et de consulter le résultat de la requête. On évite ainsi d'augmenter l'espace des actions avec les interactions, et l'espace d'états avec les autorisations accordées.

5.4.2 Modélisation du problème sous forme d'un MDP étendu

À partir d'un MDP $\langle S,A,T,r\rangle$, une action par défaut $a_{def} \in A$ (par exemple, rester sur place) et une fonction $\mathcal{R}_l$, retournant pour un couple d'état/action le niveau de restriction minimum à respecter, nous définissons un Restricted-MDP $\langle S',A',T',r',\mathcal{R}\rangle$, où :

— $S' = S \times \{\texttt{Att}\}$, où Attest le niveau d'attention de l'opérateur. Ce niveau d'attention permet de connaître le niveau d'autonomie de l'agent.
— $A' = A \cup \texttt{TO}$, où TOest l'action de télé-opération. TOpermet d'indiquer que ce n'est pas l'agent qui décide de l'action, mais l'opérateur. Soit $a_{def} \in A'$ l'action à effectuer par défaut en cas de requête non acceptée.
— $r'(s,a) = r(s) + c(\mathcal{R}(s,a))$, la fonction de récompense initiale additionnée au coût de sollicitation de l'opérateur.
— $T'((s,l),a,(s',l'))$ tel que définie par l'algorithme 8. De manière générale, en fonction de la restriction requise, nous multiplions la probabilité que l'opérateur accepte (ou refuse) la requête, par celle que l'agent atteigne l'état suivant à partir de l'action autorisée (ou l'action par défaut si refusée).

Algorithme 8 : T'((s,l),a,(s', l'))

```
si l=BUSY and l'≠l alors retourner 0 ;
si a = TO alors
    si l = BUSY alors retourner T(s,a_def,s') ;
    suivant l' faire
        cas où PILOT faire retourner P_TO(s,l) Σ_{a∈A} π_TO(s,a)T(s,a,s') ;
        cas où BUSY faire retourner (1 − P_TO(s,l))T(s,a_def,s') ;
        autres cas faire retourner 0 ;
suivant R(s,a) faire
    cas où FREE faire
        si l = l' alors retourner T(s,a,s');
        sinon retourner 0;
    cas où WATCH_REQUIRED faire
        suivant l' faire
            cas où BUSY faire retourner (1 − P_w(s,a,l)T(s,a_def,s');
            cas où WATCH faire retourner P_w(s,a,l)T(s,a,s');
            autres cas faire retourner 0;
    cas où REQUEST_REQUIRED faire
        si l = l' alors retourner (1 − P_OK(s,a,l))T(s,a_def,s') + P_OK(s,a,l)T(s,a,s');
        retourner 0;
autres cas faire retourner 0;
```

Pour trouver une politique, nous résolvons l'équation 5.1, qui est une version adaptée de l'équation de Bellman. Elle différencie les cas où l'agent est autonome et où l'opérateur pilote l'agent. Dans le premier cas, l'utilité correspond à la valeur associée à l'action qui maximisera l'utilité obtenue. Dans l'autre cas, l'utilité est celle de l'action choisie par l'opérateur. Comme l'agent n'a pas de certitude sur cette action choisi, l'utilité est calculée en faisant la moyenne de l'utilité des actions choisies, pondérée par la probabilité pour chaque action qu'elle soit choisie par l'opérateur. Cette équation est résolue à partir de Value Iteration, adapté de sorte à obtenir

l'algorithme 9. Ainsi, lorsque l'opérateur télé-opère le robot, l'agent lui laisse le contrôle.

$$\forall(s,l) \in S', V((s,l)) = \begin{cases} \max\limits_{a \in A, a_a \in A_a} r'((s,l),a) + \sum\limits_{(s',l') \in S'} T'((s,l),a,(s',l'))V((s',l')) & \textit{pour } l \neq \texttt{PILOT} \\ \sum\limits_{a \in A} \pi_{\texttt{TO}}(s,a) \times \left(r'((s,l),a) + \sum\limits_{(s',l') \in S'} T'((s,l),a,(s',l'))V((s',l')) \right) & \textit{pour } l = \texttt{PILOT} \end{cases} \tag{5.1}$$

:

Algorithme 9 : Adaptation de Value Iteration

Données : MDP $\langle S,A,T,r \rangle$, facteur d'atténuation γ, fonction $\mathcal{R}_l$

`/* Initialisation */`

$S' \rightarrow S \times \{\texttt{BUSY},\texttt{READY},\texttt{WATCH},\texttt{PILOT}\}$;
$A' \rightarrow A \cup \{\texttt{TO}\}$;
$r'(s,a) = r(s) + c(\mathcal{R}(s,a))$;
pour $ss \in S'$ **faire**
  $V(s) = 0$;

`/* Planification */`

tant que $\langle Condition \rangle$ **faire**
  pour $s_t \in S$ **faire**
    pour $l \in \{\textit{BUSY},\textit{READY},\textit{WATCH},\textit{PILOT}\}$ **faire**
      $ss_t|(s,l)$ **si** $l \neq \textit{PILOT}$ **alors**
        $V'(ss_t) \leftarrow \max_{a \in A} \left(r(ss_t,a) + \gamma \sum_{ss_{tt} \in S'} T(ss_t,a,ss_{tt})V(ss_{tt}) \right)$;
        $\pi(ss_t) \leftarrow \text{argmax}_{a \in A} \left(r'(ss_t,a) + \gamma \sum_{ss_{tt} \in S'} T'(ss_t,a,ss_{tt})V(ss_{tt}) \right)$;
      sinon
        $V'(ss_t) \leftarrow \sum_{a \in A} \pi_{\texttt{TO}}(s,a) \left(r'(ss_t,a) + \gamma \sum_{ss_{tt} \in S'} T'(ss_t,a,ss_{tt})V(ss_{tt}) \right)$;
        $\pi(ss_t) \leftarrow \texttt{TO}$;
  $V \leftarrow V'$;
retourner V,π;

Ce modèle permet de respecter les restrictions imposées par l'opérateur. Seulement, celui-ci présente plusieurs limites.

En effet, comme les contraintes sont exprimées dans la fonction de transition, celles-ci ne sont effectuées, lors de l'exécution, que lorsque l'agent atteint l'état restreint, donc au dernier moment. Dans cette situation, l'opérateur n'est pas en mesure d'analyser la situation avant de devoir fournir une réponse.

De plus, nous avons pu observer au cours des tests de l'algorithme qu'après un refus, l'agent, effectuant l'action par défaut (rester sur place), ne change pas d'état, et effectue donc la même action, dictée par la politique. Il en résulte que l'agent, une fois qu'il demande une autorisation, la redemande jusqu'à ce que l'opérateur accepte.

Enfin, l'agent n'est pas en mesure de faire la moindre requête à l'opérateur si celui-ci pilote le robot. Il ne peut pas non plus agir en même temps qu'interagir avec l'opérateur.

5.5 Amélioration du Restricted-MDP en $\mathfrak{R}$-MDP

Pour corriger le premier problème, nous proposons d'ajouter à la planification les actions associées à l'autonomie ajustable. Nous nous inspirons des travaux dans [Mouaddib *et al.*, 2010a] pour gérer le niveau d'autonomie, afin de permettre à l'agent de modifier son niveau d'autonomie tout en continuant à suivre sa politique courante. Ainsi, l'agent continue d'agir en même temps pendant qu'il interagit avec l'opérateur.

5.5.1 Description du modèle

À partir d'un MDP $\langle S,A,T,r\rangle$, d'une fonction $\mathcal{R}$ listant les restrictions, et $\mathcal{R}_l$, retournant le niveau d'une restriction, nous définissons un $\mathfrak{R}$-MDP $\langle S',A',T',r,\mathcal{R},\mathcal{R}_l\rangle$ où :

Espace d'état

En prenant pour inspiration les travaux de [Mouaddib *et al.*, 2010a], nous incluons dans l'espace d'état le niveau d'attention souhaité par l'opérateur. Il faut également y inclure les autorisations accordées (ou rejetées) par l'opérateur. Pour éviter de multiplier par 3^{n_a}(accepté, refusé, aucune demande) le nombre d'états, rendant rapidement la politique difficilement calculable, nous proposons de nous limiter à l'enregistrement de la dernière requête acceptée. Ainsi, nous définissons une représentation factorisée de l'espace d'état suivant :

$$S' = S \times \{\texttt{Att}\}^2 \times \{auth\}$$

où :
- S est prédéfini dans le modèle initial.
- `Att`représente le niveau d'attention de l'opérateur. Ici deux sont renseignés.
  1. l_r, le niveau d'attention demandé par l'agent.
  2. l_o, le niveau d'attention de l'opérateur.
- $auth$ représente la dernière autorisation envoyée par l'opérateur avant que l'action autorisée ne soit faite.

Pour simplifier les notations, nous déclarons les fonction suivantes pour extraire les composantes d'un état factorisé :
- $s(ss)$ est composante $s \in S$ d'un état $ss \in S'$
- $l_r(ss)$ est composante l_r d'un état $ss \in S'$
- $l_o(ss)$ est composante l_o d'un état $ss \in S'$
- $auth(ss)$ est composante $auth$ d'un état $ss \in S'$

Espace d'actions

Nous augmentons l'espace d'actions pour y inclure les actions d'interactions avec l'opérateur pour changer le niveau d'autonomie, donc le niveau d'attention de l'opérateur et donc l'état factorisé. D'une part les actions exprimant l'intention de l'agent, et d'autre par la prise de contrôle de l'opérateur.

$$A' = \begin{cases} A \\ \times A_a = \{+,-,\varnothing\} \cup \{req_i\}_{\forall i \in [1..n_a]} \\ \times A_o = \{\texttt{BUSY},\texttt{READY},\texttt{WATCH},\texttt{PILOT},\varnothing\} \end{cases}$$

- A : liste d'action pré-définie dans le MDP initial.

— A_a : liste d'actions de l'agent pour modifier son autonomie (l'augmenter, la diminuer, ne rien faire ou demander une autorisation).
— req_i : demande d'autorisation pour la restriction numéro i.
— A_o Action de l'opérateur pour varier l'autonomie. Celui-ci peut directement choisir de piloter, de surveiller ou de se déclarer indisponible (changement du niveau d'attention).
— $\varnothing$ action nulle (ne rien faire).

À chaque pas de temps, une action pour chaque est choisi, permettant ainsi à l'agent de contrôler le robot tout en soumettant des requêtes à l'opérateur.

Fonction de récompense

La fonction de récompense doit être modifiée, car elle ne doit plus prendre en compte le niveau d'attention effectif de l'opérateur, mais le niveau d'attention demandé par l'agent. En effet, si l'opérateur décide de lui même de changer son niveau d'attention, il ne devrait pas y avoir d'impact sur le robot. D'un autre côté, demander de l'aide à l'opérateur est gênant même si l'opérateur refuse d'accorder cet aide.

$$r'((s,l_r,l_o),a,a_a) = \begin{cases} r(s,a) & \text{Coût initial du MDP} \\ +c(l_r) & \text{Coût de la demande d'attention} \\ +c(\texttt{REQUEST\_REQUIRED}) & \text{Si } a_a \in \{req_i\}_{\forall i \in [1..n_a]} \end{cases}$$

La fonction de récompense est modifiée pour inclure le coût d'une interaction avec l'opérateur. La troisième ligne correspond au coût d'une demande d'autorisation.

Fonction de transition

La fonction de transition doit être mise à jour. En effet, elle dépend de 3 actions simultanées : le contrôle du robot, les requêtes de l'agent et le changement d'attention de l'opérateur. Il faut alors gérer les contrôles du robot, ce que faisait la fonction de transition du MDP initial, mais aussi le changement de niveau d'attention requis, en fonction des interactions initiées par l'agent. Il faut aussi changer le niveau d'attention observé de l'opérateur en fonction des actions des interactions initiées par l'opérateur.

$$\begin{aligned} &T'(\langle s,l_r,l_o,auth\rangle,a,a_a,a_o,\langle s',l'_r,l'_o,auth'\rangle) = \\ &\quad T_r(\langle s,l_o,auth\rangle,a,s') && \text{Mouvement du robot} \\ &\quad *T_{auth}(\langle s,l_o,auth\rangle,a,a_a,auth') && \text{Gestion demande autorisations} \\ &\quad *T_a(l_r,a_a l'_r) && \text{Interaction agent} \rightarrow \text{opérateur} \\ &\quad *T_o(l_o,a_o,l'_o) && \text{Interaction opérateur} \rightarrow \text{agent} \end{aligned}$$

La fonction de transition est composée de celle définissant l'impact du contrôle du robot sur ses mouvements(algorithme 10), modifiée afin d'assurer qu'aucune action exécutée soit contraire aux actions autorisées de l'agent. Elle est couplée à la fonction de transition associée à la demande d'autorisation (algorithme 11), assurant la gestion de la dernière autorisation donnée et sa réinitialisation une fois l'action accomplie. Elle est enfin combinée aux fonctions de transitions associées aux changements de niveau d'attention demandé (équation 5.2) et effectif (équation 5.3).

Algorithme 10 : $T_r(\langle s,l_o,auth\rangle,a,s')$

```
1 ℜ = R(s,a);
2 si RL_ℜ > l_o or n(ℜ) = 0 or n(ℜ) = auth alors
3     retourner T(s,a,s')
4 sinon retourner T(s,π_ℜ(s),s');
```

Algorithme 11 : $T_{auth}(\langle s,l_o,auth\rangle,a,a_a,auth')$

```
1  suivant a_a faire
2      cas où req_i faire
           // L'agent demande une autorisation
3          rep_ok ← P_OK(i,l_o);
4          si auth ≠ auth' alors
5              si auth' = i alors retourner rep_ok;
6              si R_a(s,a) = auth and auth' = 0 alors retourner 1 − rep_ok;
7              retourner 0
8          si i = auth alors
9              si R_a(s,a) = auth alors retourner rep_ok;
10             retourner 1;
11         retourner 1-rep_ok;
12     otherwise do
13         si R_a(s,a) = auth et auth' = 0 alors retourner 1 ;
14         si auth = auth' alors retourner 1;
15         retourner 0;
```

$$T_a(l_r,a_a,{l_r}') = \begin{cases} & si\ a_a = +\ et\ {l_r}' = l_r + 1 \\ 1 & ou\ a_a = -\ et\ {l_r}' = l_r - 1 \\ & ou\ a_a \notin \{+,-\}\ et\ l_r = {l_r}' \\ \\ 0 & sinon \end{cases} \tag{5.2}$$

$$T_o(l_o,a_o,{l_o}') = \begin{cases} 1 & \text{Si } a_o \text{ et } l'_o \text{ correspondent} \\ 0 & sinon \end{cases} \tag{5.3}$$

5.5.2 Résolution

Prédiction

Afin de déterminer les chances qu'un changement de transition soit accordé, nous définissons $\pi_o(s,a_o)$ la probabilité pour l'opérateur d'accomplir l'action $a_o \in A_o$ dans l'état $s \in S'$. Celle-ci est définie à partir des connaissances sur l'opérateur en section 5.3. Exemple pour une demande de surveillance de l'agent (tous les cas non décrits ont une probabilité nulle) :

$$\pi_o((s,\texttt{WATCH},\texttt{READY}),\texttt{WATCH}) = P_w(s,a,l_o)$$
$$\pi_o((s,\texttt{WATCH},\texttt{READY}),\texttt{BUSY}) = 1 - P_w(s,a,l_o)$$
$$\pi_o((s,\texttt{WATCH},\texttt{PILOT}),\texttt{WATCH}) = 1$$

Réductions

Pour simplifier la lecture, nous proposons des réductions de noms :
— $T'(\langle s,l_r,l_o,auth\rangle,a,a_a,a_o,\langle s',l'_r,l'_o,auth'\rangle) = T_{a,a_a}$
— $r'((s,l_r,l_o),a,a_a) = r_a$

Fonction de valeur

$$V(s_o \equiv \langle s,l_r,l_o,auth\rangle) =$$
$$\begin{cases} \max\limits_{a\in A, a_a\in A_a} r_a + \sum\limits_{s'_o\in S', a_o\in A_o} T_a V(s'_o) & pour\ l_o \neq \texttt{PILOT} \\ \max\limits_{a_a\in A_a} \sum\limits_{a\in A} \pi_{\texttt{TO}}(s,a) \left[r_a + \sum\limits_{s'_o\in S', a_o\in A_o} T_a V(s'_o) \right] & pour\ l_o = \texttt{PILOT} \end{cases} \quad (5.4)$$

Une fois encore, l'équation utilisée pour calculer l'utilité diffère en fonction du niveau d'attention de l'opérateur. S'il pilote l'agent, alors nous considérons l'ensemble des actions possibles dans A, pondérés par la probabilité qu'elle soit sélectionnée lors de la télé-opération.

Notre deuxième solution permet d'assurer que l'agent demande à l'avance les autorisations ou les changements d'attention de l'opérateur. Notamment, l'agent ne peut plus demander de changement brutal d'autonomie.

Néanmoins, il ne résout pas le problème de répétitions des autorisations. Nous pourrions envisager un apprentissage des fonctions P_{OK} en fonction des réponses reçues, permettant ainsi à l'agent, après re-planification, de calculer une stratégie différente si les chances de réponses positives sont trop faibles. Néanmoins, cela ne correspondrait pas au comportement souhaité en section 5.2.

Pour nous rapprocher du comportement souhaité, nous présentons dans la section suivante une solution calculant une politique en fonction des actions déjà interdites.

5.6 Une politique pour chaque refus

La gestion des requêtes interdites est un vrai problème car elle sous-entend retenir le résultat des précédentes. Nous pourrions calculer une politique pour chaque ensemble de requêtes refusées, néanmoins cette solution implique un grand nombre de calculs, pour des situations qui ne devraient pas arriver. Nous proposons ci-dessous un algorithme permettant de limiter le nombre de cas d'interdictions étudiées à ceux potentiellement atteignables avec la politique calculée au fur et à mesure de la planification.

5.6.1 Algorithme

Soit A_f la liste de requêtes déjà refusées, π_{A_f} la politique de l'agent en considérant l'interdiction de faire les actions dans A_f, et V_{A_f} sa fonction de valeur correspondante. Pour limiter

le nombre de cas traités, nous proposons l'algorithme 14, permettant de calculer la politique associée à un ensemble de requêtes refusées possibles.

L'algorithme traite le cas de la télé-opération en ligne 11, et les autres cas en ligne 22. Dans le premier cas, l'algorithme ne cherche qu'à évaluer la politique de que l'opérateur suit, permettant de savoir s'il est plus intéressant de laisser l'opérateur faire, car il ne sera pas limité par les restrictions imposées à l'agent. Dans le second cas, nous cherchons à évaluer l'utilité de chaque action pour choisir la plus utile.

La gestion des demandes d'autorisation se fait aux lignes 15 et 26. La fonction de valeur calculée dépend à la fois du cas où la permission est accordée ou rejetée. On distingue alors ces deux cas dans la fonction de valeur en ligne 18 et 29. Pour cela, nous considérons les états atteignables quand la demande est acceptée (S_{OK}) des états où la demande est refusée ($S_{\neg OK}$). La fonction de transition T_{auth}, associée à une demande, ne distingue pas ces deux situations, il faut donc en définir une, T_{OK} dans le cas d'un accord (algorithme 12), et $T_{\neg OK}$ dans le cas d'un désaccord (algorithme 13). Il suffit alors, dans la fonction T_{a,a_a} de remplacer la composante T_{auth} par la fonction T_{OK}, ou $T_{\neg OK}$, en fonction de la réponse de l'opérateur. On appelle ces fonctions respectivement $T_{a,a_a,OK}$ et $T_{a,a_a,\neg OK}$.

Enfin, en ligne 34, si une demande de permission est considérée optimale, alors le cas de son refus est ajouté à la planification, combinée aux autorisations déjà refusées.

Algorithme 12 : $T_{OK}(\langle s,auth,l_o\rangle,a,a_a,auth')$

```
1 suivant a_a faire
2     cas où Req_i faire
3         si auth' = i alors retourner P_OK(i,l_o);
          /* Autre cas impossible car la transition suppose l'accord   */
4         sinon retourner 0;
          /* Si l'action n'est de type requête, utiliser la fonction de
             transition classique.                                     */
5     autres cas faire
6         retourner T_auth(⟨s,auth,l_o⟩,a,a_a,auth')
```

Algorithme 13 : $T_{\neg OK}(\langle s,auth,l_o\rangle,a,a_a,auth')$

```
1 suivant a_a faire
2     cas où Req_i faire
          /* Pas de changement d'autorisation ou fin d'autorisation dans un
             cas de refus                                              */
3         si auth' = auth ou (auth' = 0 et R_l(s,a) = auth) alors
4             retourner 1 − P_OK(i,l_o)
5         retourner 0;
          /* Si l'action n'est de type requête, utiliser la fonction de
             transition classique.                                     */
6     autres cas faire
7         retourner T_auth(⟨s,auth,l_o⟩,a,a_a,auth')
```

Algorithme 14 : Value Iteration adapté à l'autonomie ajustable

initialiser $V^*(s,l_r,l_o,auth) = r_a \forall s \in S, l_r \in Att, l_o \in Arr, auth \in [0..n_a]$;
$A_{ff} = \{\emptyset\}$;
tant que <*Condition*> **faire**
  pour $A_f \in A_{ff}$ **faire**
    pour $a_a \in A_a \setminus A_f$ **faire**
      initialiser, si nécessaire, $V^*_{A_f \cup \{a_a\}}(s,l_r,l_o,auth) = \max_{a,a_a \in A \times A_a \setminus (A_f \cup \{a_a\})} r_a$;
    pour $s,l_r,l_o,auth \in S \times \texttt{Att} \times \texttt{Att} \times [1..n_a]$ **faire**
      $ss \leftarrow \langle s,l_r,l_o,auth \rangle$;
      $a^*,a_a^* \leftarrow \varnothing$;
      ${V_{A_f}}'^*(ss) \leftarrow -\infty$;
      si $l_o = \texttt{PILOT}$ **alors**
        pour $a_a \in A_a$ **faire**
          si $a_a \notin \{req_i\}_{\forall i \in [1..n_a]}$ **alors**
            ${V_{A_f}}'(ss) \leftarrow sum_{a \in A} \pi_{\texttt{TO}}(ss,a) \times \left[r_a + \sum_{s' \in S'} T_{a,a_a}.V_{A_f}(s') \right]$;
          sinon
            $S_{OK} \leftarrow \{s' \in S' | auth(s') = 0\}$;
            $S_{\neg OK} \leftarrow \{s' \in S' | auth(s') = auth\}$;
            ${V_{A_f}}'(ss) \leftarrow \sum_{a \in A} \pi_{\texttt{TO}}(ss,a) \times$
              $\left[r_a + \sum_{s' \in S_{\neg OK}} T_{a,a_a,\neg OK}.V_{A_f \cup \{a_a\}}(s') + \sum_{s' \in S_{OK}} T_{a,a_a,OK}.V_{\emptyset}(s') \right)\Big]$;
          si ${V_{A_f}}'(ss) > {V_{A_f}}'^*(ss)$ **alors**
            ${V_{A_f}}'^*(ss) \leftarrow {V_{A_f}}'(ss)$;
            $a^*,a_a^* \leftarrow \max_a \pi_{\texttt{TO}}(ss,a),a_a$;
      sinon
        pour $a,a_a \in A \times A_a \setminus A_f$ **faire**
          si $a_a \notin \{req_i\}_{\forall i \in [1..n_a]}$ **alors**
            ${V_{A_f}}'(ss) \leftarrow r_a + \sum_{s' \in S'} T_{a,a_a}.V_{A_f}(s')$;
          sinon
            $S_{OK} \leftarrow \{s' \in S' | auth(s') = 0\}$;
            $S_{\neg OK} \leftarrow \{s' \in S' | auth(s') = auth\}$;
            ${V_{A_f}}'(ss) \leftarrow$
              $r_a + \sum_{s' \in S_{\neg OK}} T_{a,a_a,\neg OK}.V_{A_f \cup \{a_a\}}(s') + \sum_{s' \in S_{OK}} T_{a,a_a,OK}.V_{\emptyset}(s'))$;
          si ${V_{A_f}}'(ss) > {V_{A_f}}'^*(ss)$ **alors**
            ${V_{A_f}}'^*(ss) \leftarrow {V_{A_f}}'(ss)$;
            $a^*,a_a^* \leftarrow a,a_a$;
      $\pi^*_{A_f}(ss) \leftarrow a^*,a_a^*$;
      si $a_a^* \in \{req_i\}_{\forall i \in [1..n_a]}$ *and* $A_f \cup \{a_a^*\} \notin A_{ff}$ **alors**
        $A_{ff} \leftarrow A_{ff} \cup \{A_f \cup \{a_a^*\}\}$;
  $V \leftarrow V'^*$;
retourner π^*

Au cours de l'exécution, il suffira à l'agent de stocker en dehors de l'état les action déjà interdites, et de choisir la politique à partir de cette liste.

5.6.2 Améliorations possibles

Cet algorithme, théoriquement, devrait assurer nos souhaits exprimés en section 5.2 quant au comportement de l'agent. Néanmoins, celui-ci reste assez lourd, notamment du fait que l'espace d'état est riche. Considérer l'ensemble de cet espace d'état comme état suivant possible est excessif, surtout en connaissant les règles associées aux changements d'attention de l'opérateur ou du souhait d'autonomie de l'agent. Calculer un sous-ensemble d'états de successeurs permettrait de diminuer le temps d'évaluation de l'utilité d'un état.

D'autre part, nous pourrions considérer l'hypothèse d'avoir un état initial au système. Autrement dit, nous aurions un état $s_0 \in S$, où l'agent se situerait. Cette information permettrait d'utiliser d'autres algorithmes, comme LAO* [Hansen et Zilberstein, 2001], permettant de calculer une politique en ne parcourant que les états atteignables. Il a néanmoins pour inconvénient de ne pas calculer une politique si un état n'est pas supposé être atteint, ce qui peut être gênant en cas d'imprévu.

5.7 Conclusion

Dans ce chapitre, nous avons défini le problème visant à intégrer des restrictions dans la politique d'un agent à autonomie ajustable, afin de le mener à changer son autonomie en fonction de la situation.

Nous avons proposé différentes méthodes servant à restreindre un agent en fonction de l'attention de son opérateur, et permettant à l'agent de varier son autonomie si nécessaire. La première permettait de gérer simplement les échanges agent/opérateur en les incluant dans la fonction de transition. La seconde permettait à l'agent de gérer indépendamment le contrôle du robot et son autonomie. Le dernier modèle permet de planifier la variation de l'autonomie pour permettre une transition douce d'attention à l'opérateur, tout en limitant la répétition des demandes.

Dans la partie expérimentation, nous étudierons la capacité d'un agent à s'adapter à certains types d'opérateur, et étudierons le taux d'occupation de l'opérateur.

Conclusion de la Partie 2

L'objectif de cette thèse est de permettre à un système multi-agents de pouvoir s'appuyer sur des opérateurs pour lui permettre une meilleure adaptation à l'imprévu, que ce soit pour gérer des autorisations d'accès ou de la télé-opération en cas de nécessité. Nous nous sommes intéressés pour cela à deux problématiques. La première concerne l'estimation de la politique d'un agent télé-opéré pour qu'un agent autonome puisse s'y coordonner. La seconde consiste à permettre à un agent autonome de respecter des restrictions imposées par l'opérateur, et de modifier son niveau d'autonomie si ces restrictions l'exigent.

Estimation d'une politique

Nous avons expliqué dans le chapitre 4 la difficulté pour un agent autonome de se coordonner avec un agent télé-opéré. En effet, les décisions sont prises par un humain, qui n'a pas les mêmes perceptions de l'utilité d'une action, de son environnement et est sujet à des perturbations telles que le stress, ou encore à cause de sa faculté d'apprentissage au cours d'une mission. Tous ces facteurs font qu'il est difficile pour un agent autonome d'estimer la politique d'un agent télé-opéré, et donc de s'y synchroniser.

Pour répondre à ce problème, nous avons d'abord proposé 3 méthodes basées sur l'apprentissage pour permettre à l'agent autonome d'estimer la politique d'un agent télé-opéré. La première, `FORCE`, se base sur l'apprentissage des dernières actions sélectionnées par l'opérateur pour chaque état, afin d'adapter la politique des autres états de sorte à correspondre à ces informations. La seconde, `LEARN`, a pour principe d'apprendre les préférences de l'opérateur sur les états déduits des actions qu'il a choisi. La dernière, `DIST`, est similaire à `LEARN`, mais avec une généralisation de l'apprentissage sur les préférences des états réduites. Nous avons également proposé des améliorations de ces méthodes suite à des comportements imprévus rencontrés au cours des expériences telles que l'apparition d'états plus intéressants que l'objectif, selon le modèle des préférences de l'opérateur sur les états.

Nous développerons dans le chapitre 6 les expériences ainsi que l'évaluation des modèles qui ont été présentés au chapitre 4.

Autonomie ajustable

Dans le chapitre 5, nous avons commencé par présenter le comportement souhaité d'un agent autonome. Il faut qu'il soit capable de respecter des restrictions telles qu'une interdiction d'entrer dans une zone sans surveillance d'un opérateur. Il lui faut donc être capable de demander l'assistance de l'opérateur, mais de s'adapter en cas indisponibilité.

Pour répondre à ce problème, nous avons d'abord présenté une formalisation des restrictions, des niveaux d'attention de l'opérateur et des fonctions nécessaires à l'agent pour interagir

avec l'opérateur, telle qu'une fonction d'attente de télé-opération. Nous avons ensuite présenté un premier modèle, intégrant les interactions entre l'agent et l'opérateur directement dans la fonction de transition. Néanmoins, cette méthode avait des inconvénients, comme l'impossibilité d'anticiper une demande d'assistance à l'opérateur, ou l'incapacité de l'agent à redemander le contrôle une fois qu'il n'a plus besoin de la télé-opération. Nous avons alors modifié le modèle pour y intégrer l'opérateur et se rapprocher d'un modèle à initiative mixte, permettant ainsi de planifier les interactions avec l'opérateur indépendamment des actions de contrôle du robot. Enfin, pour éviter de demander de manière répétée une autorisation qui aurait été refusée, nous avons proposé une variante du second modèle enregistrant dans l'état du système les dernières autorisations refusées.

Dans le chapitre 7, nous étudierons la capacité d'un agent à s'adapter à certains types d'opérateur, et le taux d'occupation de l'opérateur.

Troisième partie

Evaluation expérimentale

Chapitre 6

Résultats expérimentaux sur l'estimation de la politique d'agents à politique non-optimale

Sommaire

Dans ce chapitre, nous allons d'évaluer l'efficacité et la stabilité des méthodes `FORCE`, `LEARN`, `DIST`, introduites en section 4.2, ainsi que leurs méthodes dérivées, issues des améliorations décrites en section 4.3. Pour cela, nous présenterons nos expérimentations concernant la prédiction de la politique d'un agent non-optimal. Dans un premier temps, nous étudions différents modèles pour comparer nos méthodes d'apprentissage, dont une méthode d'apprentissage supervisée, une méthode d'apprentissage par renfocrement inverse, et une méthode d'apprentissage statistique. Nous décrirons par la suite nos conditions d'expériences, et enfin nous présenterons les résultats obtenus au cours de celles-ci. Nous conclurons à partir de ces résultats sur l'efficacité et la stabilité de nos méthodes de prédictions.

6.1 Premier jeu d'expériences

Un premier jeu d'expérience, présenté à la conférence ECTA [Lelerre et Mouaddib, 2016], avait pour but d'introduire les concepts de `FORCE`, `LEARN` et la première version de `DIST`, et de

tester leur efficacité et leur stabilité. Pour cela, nous avons dans un premier temps un scénario d'expériences, puis présenté un algorithme de comparaison afin de situer nos travaux avec l'état de l'art. Enfin, nous avons expérimenté et comparé les résultats obtenus.

6.1.1 Description de l'expérience

Scénario

Notre expérience avait pour objectif de tester l'efficacité de la prédiction de la politique d'un agent télé-opéré. Pour cela, nous avons considéré un agent dans un labyrinthe en 2 dimensions. Cet environnement a une propriété correspondant à l'éclairage de chaque position. L'agent est dans un état initial, et doit arriver dans un état objectif. L'agent doit rejoindre son objectif en maintenant un niveau d'éclairage correspondant au niveau d'éclairage à la position du robot.

L'agent est animé par une politique autonome. Celle-ci est néanmoins modifiée pour simuler les préférences et les hésitations de l'opérateur. Pour les préférences, la politique a été fixée à certains états pour forcer l'agent à emprunter un chemin différent de l'optimal, tandis que d'autres états étaient interdits dans le calcul de la politique. Les hésitations ont été modélisées par des politiques stochastiques à des endroits fixes de la politique.

L'objectif de ce scénario est d'évaluer les performances de prédiction des modèles `FORCE`, `DIST` et `LEARN`. Pour cela, ces méthodes sont initialisées avec une politique optimale, mais sans connaissances sur les variations apportées pour simuler les critères de l'opérateur. Durant l'exécution, les états atteints et les actions observées sont envoyés dans ces modèles pour mettre à jour la politique en ligne. Pour évaluer les performances de ces politiques, nous comparons le nombre de fois où chaque méthode de prédiction s'est trompé d'action. Comme nous considérons des missions de patrouilles comme application possible dans le cadre du projet DGA/ANR GARDES [3], nous avons évalué l'apprentissage des méthodes sur une cinquantaine d'exécutions de la mission, en considérant que la patrouille est une tâche à répéter. Au cours des différentes missions, seul les hésitations amènent le robot à changer de politique.

Dans un premier temps, nous avons étudié une carte de type labyrinthe, constituée principalement de couloirs. Par la suite, nous avons étudié un problème plus ouvert, constitué de quelques obstacles mais ayant un facteur de branchement plus élevé entre les états.

Modèle

Un état de l'environnement est composé des coordonnées de l'agent, et du niveau d'éclairage de ce dernier. Le niveau d'éclairage de chaque position possible de l'agent est déterminé hors de l'espace d'état.

En terme d'actions, l'agent est en mesure d'avancer d'une case dans toutes les directions, ou de changer de mode d'éclairage parmi 3 niveaux : nul, faible et élevé, correspondant chacun aux différents niveaux d'éclairage possibles de chaque case de la cartes.

Dans ce modèle, la fonction d'utilité de l'agent est définie de la manière suivante : l'agent a un coût faible à chaque action, augmentée si l'agent ne respecte pas le niveau d'éclairage à sa position. Il reçoit également une récompense en atteignant l'objectif.

Deux types de transitions sont étudiées Dans le cadre déterministe, la direction de l'agent est toujours respectée. Dans le cadre stochastique, l'impact des actions de l'agent est moins certaine, et ce dernier a des chances d'avancer dans une autre direction que celle souhaitée, indépendamment de l'hésitation de l'opérateur.

3. `https://gardes.greyc.fr/`

Pour comparer nos méthodes, nous avons cherché un modèle permettant d'apprendre la politique de l'opérateur. Pour cela, nous nous sommes tournés vers TAMER, proposée par [Knox et Stone, 2008].

Nous avons besoin du modèle d'un opérateur pour estimer la politique qu'il va appliquer à l'agent. Celle-ci est définie à partir de la politique optimale de l'agent.

Nous avons modélisé le comportement de l'opérateur à estimer avec un MDP. Pour modéliser ses préférences et les différences de perceptions avec un robot, nous avons intégré des états interdits, c'est à dire que l'opérateur ne peut choisir des actions menant vers ces états avec une forte probabilité. Nous avons également fixé la politique avant la planification à certains endroits pour imposer une trajectoire.

Afin de modéliser l'hésitation, variant en fonction de l'expertise de l'opérateur quant à la mission, nous modifions la politique à certains états sources d'hésitation pour l'opérateur, en fonction d'un paramètre $h \in [0..1]$, pour le faire choisir entre deux actions aléatoirement. Pour $h = 0$, c'est à dire sans hésitation, celui-ci choisira l'action qu'il estimera optimale, en fonction de ses préférences. Pour $h = 0.3$, c'est à dire 30% d'hésitations l'opérateur aura 70% de chances de choisir l'action optimale, et 30% un autre action.

6.1.2 Training an Agent Manually via Evaluative Reinforcement (TAMER)

Définition de TAMER

TAMER est une méthode d'apprentissage supervisée, proposée par [Knox et Stone, 2008]. Elle consiste à apprendre, dans un environnement déterministe, une politique à partir de signaux H, positifs ou négatifs, envoyés par l'opérateur à l'agent. Ainsi, l'agent planifiera ses actions afin de maximiser les signaux H reçus.

L'environnement est décrit dans ce modèle comme étant un espace d'état, où un état est un vecteur de propriétés (par exemple la position du robot dans son environnement). Par la suite, en fonction des actions effectuées, et des signaux reçues, une pondération est mise à jour pour chaque propriété de l'état. Cette dernière permet ainsi de calculer une fonction d'utilité $\hat{H}(s,a)$, correspondant à l'espérance du signal reçu en effectuant une action a dans un état s. Celle-ci est décrite en équation 6.1, où $\overrightarrow{\omega}$ est le vecteur de pondération des propriétés de l'état et $\overrightarrow{s_t}$ est le vecteur de propriétés de s_t, et $T(s_t,a)$ est une fonction de transition déterministe retournant l'état atteint s_{t+1} après avoir effectué l'action a dans l'état s_t. $\overrightarrow{\Delta}$ correspond à la différence de vecteur de deux états, permettant d'évaluer l'impact d'une action sur les paramètres du système.

$$\hat{H}(s_t,a) = \sum_i w_i.\Delta_i, \qquad s_{t+1} = T(s_t,a) \tag{6.1}$$

Au cours de l'exécution, l'agent reçoit des signaux H de la part de l'opérateur en fonction des actions qu'il a effectué. Ainsi, il met à jour le vecteur $\overrightarrow{\omega}$ via l'équation 6.2, où ω_i' correspond à la valeur du paramètre ω_i après mise à jour. α est un facteur d'apprentissage, pondérant l'influence d'un signal sur la mise à jour de la politique. . La mise à jour du vecteur $\overrightarrow{\omega}$ est donc calculée pour chaque paramètre du vecteur en comparant le signal reçu et la récompense espérée, en fonction de l'impact de ce paramètre dans cette transition selon les anciens paramètres de pondération ω_i.

$$\omega_i' = \omega_i + \alpha * (H - \sum_j \omega_j.\Delta_j).\omega_i.\Delta_i \tag{6.2}$$

A partir du signal espéré, l'action choisie par le robot correspond à celle maximisant le signal espéré à un instant donné. Celle-ci est obtenue grâce à l'équation 6.1.

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, \hat{H}(s,a) \tag{6.3}$$

Combinaison avec de l'apprentissage par renforcement

Par la suite, [Knox et Stone, 2010] et [Knox et Stone, 2012] ont proposé la méthode TAMER&RL, combinant TAMER avec une méthode d'apprentissage par renforcement afin d'améliorer les performances d'apprentissage. En effet, TAMER permet d'apprendre plus rapidement qu'avec de l'apprentissage par renforcement, comme SARSA [Sutton et Barto, 1998a], mais moins précisément. Chacune des méthodes composant TAMER&RL retournera une politique différente, il faudra choisir laquelle. Une des méthodes possibles pour ce choix consiste à l'effectuer à partir de l'équation 6.4, où β est un facteur décroissant, déterminant quelle politique suivre. Lorsque β est proche de 1, en début de mission, alors la politique principalement appliquée est celle générée par TAMER afin d'accélérer l'apprentissage. Lorsque β décroit, la politique principalement suivie est alors celle obtenue par renforcement, avec de meilleures performances.

$$P(a = \underset{a}{\operatorname{argmax}}[\hat{H}(s,a)]) = min(\beta,1) \tag{6.4}$$

Adaptation

Nous avons considéré TAMER car celui-ci vise à apprendre et à modéliser les préférences de l'opérateur quant aux actions de l'agent. Néanmoins, ces préférences sont déduites à partir de signaux envoyés par l'opérateur, alors que dans notre cas, ce dernier choisit directement les actions du robot lors de la télé-opération. Pour considérer ce cas, nous avons adapté la méthode, en envoyant un signal positif à toute action observée, que nous considérons recommandée par l'opérateur, d'une part, et un signal négatif à toute action estimée différente de celle observée, d'autre part.

Cette méthode est supposée être utilisée dans un environnement déterministe, alors que nous nous intéressons également à des environnements stochastiques. Pour adapter TAMER, nous avons modifié la fonction Δ, en prenant en compte l'ensemble des états atteignables et la probabilité de les atteindre. Elle est alors définie par l'équation 6.5.

$$\Delta = \sum_{s_{t+1} \in S} T(s_t,a,s_{t+1}) \, (\overrightarrow{s_{t+1}} - \overrightarrow{s_t}) \tag{6.5}$$

Une fois TAMER adapté à nos besoins, nous avons pu lancer les expérimentations.

6.1.3 Résultats

Nous présenterons ici les résultats obtenus en fonction du type d'environnement (ouvert ou fermé), c'est à dire du facteur de branchement, du type de transitions (déterministe ou stochastique) et de l'expertise de l'opérateur. Un opérateur inexpérimenté aura tendance à hésiter (c'est à dire choisir une action non optimale) qu'un opérateur expert. Chaque figure représentera le nombre d'erreurs rencontrées pour chaque itération de la mission.

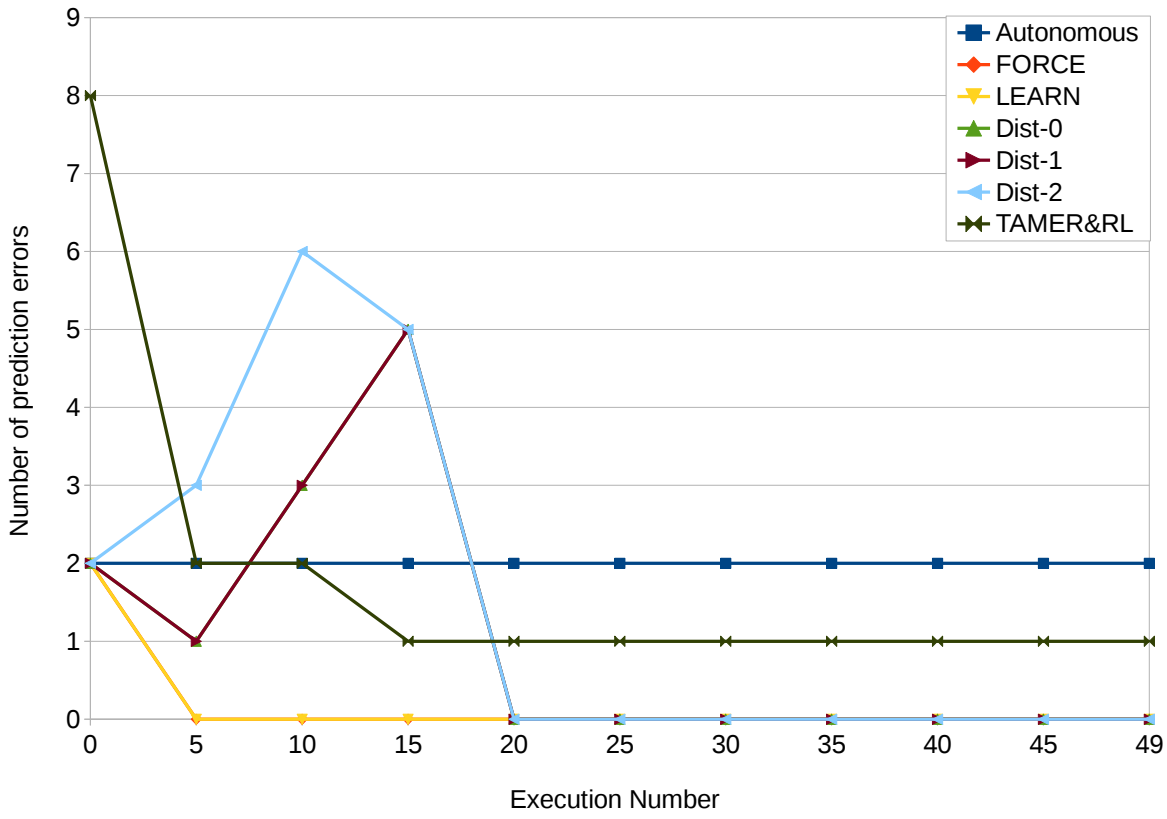


FIGURE 6.1 – Environnement fermé, déterministe et opérateur expert

Environnement fermé, déterministe et opérateur expert

Nous pouvons voir, en figure 6.1, que prédire les actions de l'opérateur à partir d'une politique optimale (courbe Autonomous) ne permet pas de prendre en compte les préférences de l'utilisateur sur la politique de l'agent.

Les politiques `FORCE` et `LEARN` sont efficaces, et permettent d'apprendre rapidement la politique de l'opérateur, contrairement à TAMER&RL qui se rapproche de la politique à estimer sans pouvoir apprendre toutes les préférences. `DIST` parvient à apprendre la politique, mais de manière beaucoup plus instable et en plus de temps.

Au cours de ces expériences, nous avons évalué la méthode `DIST` avec plusieurs valeurs de δ pour évaluer l'impact de ce paramètre. Ainsi, la courbe `DIST`-0 correspond à la méthode `DIST` paramétrée avec δ=0. Ces courbes nous montrent qu'en augmentant ce paramètre, les courbes sont plus instables.

Environnement fermé, déterministe et opérateur non expert

Lorsque l'opérateur n'est pas expert, et hésite (à 30% en figure 6.2), la prédiction est plus difficile. `FORCE` et `LEARN` restent meilleurs que les autres en ne faisant des erreurs que leors des hésitations. TAMER&RL reste stable, pour un opérateur hésitant, mais n'arrive pas à apprendre l'ensemble des préférences. Enfin, `DIST` est très instable, et plus δ (distance maximale pour la mise à jour de la récompense pour la méthode `DIST`) est grand, pire c'est. Néanmoins, avec δ petit, la méthode permet tout de même d'avoir une estimation proche de la politique observée.

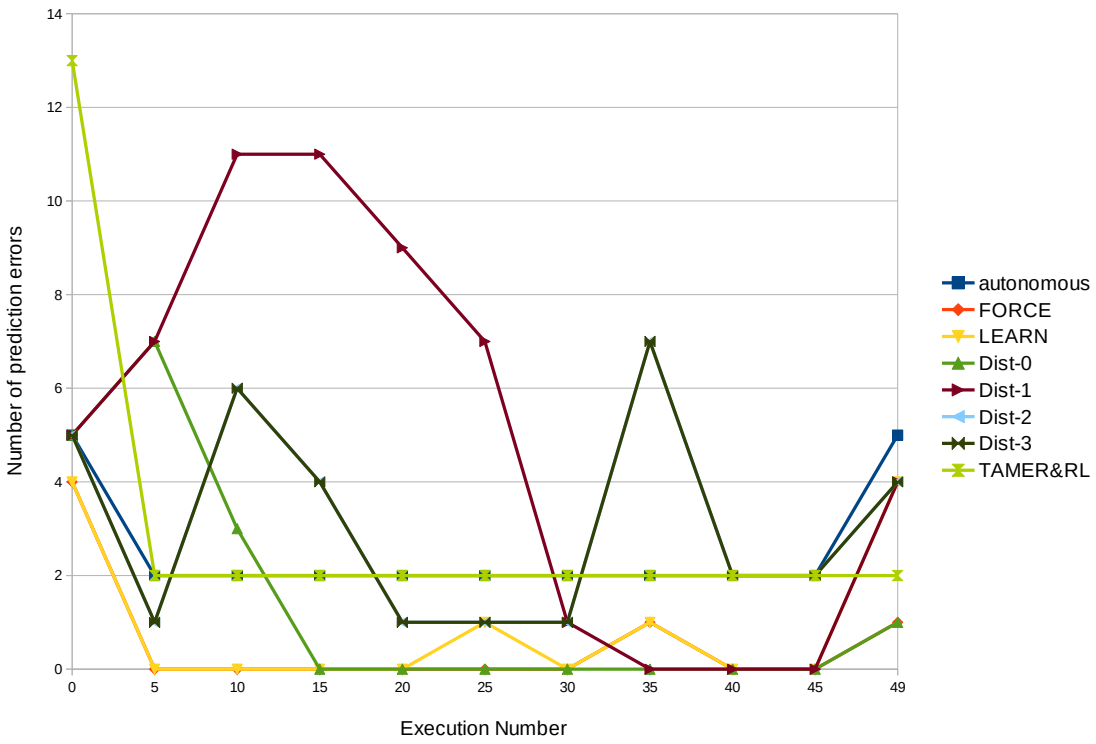


FIGURE 6.2 – Environnement fermé, déterministe et opérateur non-expert

Ceci est sans doute dû d'une part aux remarques faites en section 4.3.1, et d'autre part du fait du manque de généralisation, entraînant un apprentissage plus long.

Environnement fermé, stochastique et opérateur expert

Un environnement stochastique impacte beaucoup sur les résultats des modèles, comme le montre la figure 6.3.

D'une part, TAMER&RL perd toute stabilité et n'arrive plus à se rapprocher de la politique exécutée. Ceci est sans doute dû au fait qu'il n'est pas conçu pour un cadre stochastique. D'autre part, les méthodes `DIST` ont gagné en stabilité. Ceci est dû au fait que les transitions impliquent plus d'états, vu que plusieurs états sont atteignables à partir d'une action. Les méthodes `DIST` peuvent ainsi mieux généraliser l'apprentissage. Elles restent néanmoins plus lentes que les méthodes `LEARN` et `FORCE`.

Environnement fermé, stochastique et opérateur non expert

Dans le cas d'un opérateur non-expert avec 30% d'hésitation, nous voyons en figure 6.4 que l'ensemble des méthodes, à l'exception de `FORCE` perd en stabilité. En effet, entre les états atteints différents de la trajectoire espérée, dû aux transitions stochastiques, et les hésitations de l'opérateur, beaucoup de changement de trajectoires vis à vis de celle prévue à l'origine sont observables.

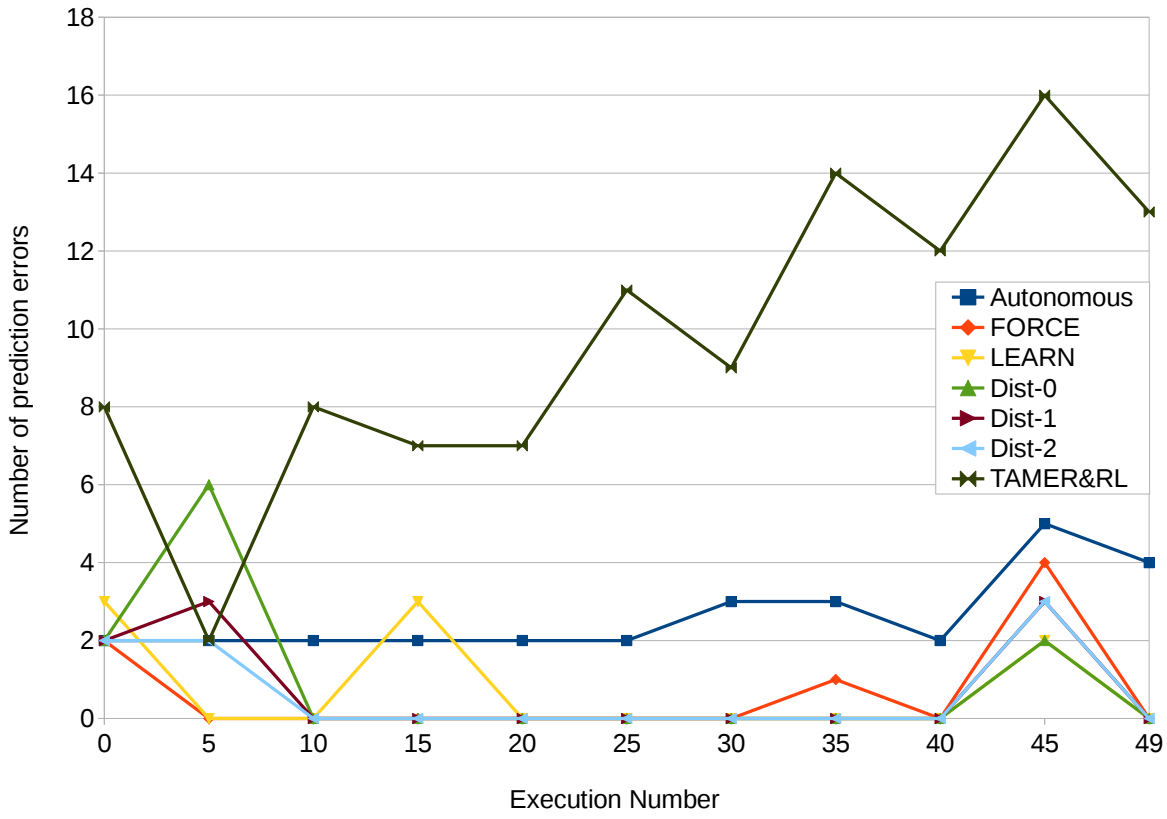


FIGURE 6.3 – Environnement fermé, stochastique et opérateur expert

Seules `FORCE` et `LEARN` permettent un apprentissage efficace, permettant de mieux prédire que la politique sans apprentissage. `DIST` est plus performant que TAMER&RL sur la durée, mais n'est pas exploitable pour autant, surtout avec peu d'apprentissage.

Environnement ouvert, déterministe et opérateur expert

L'environnement ouvert correspond à un espace avec moins d'obstacle, aucun couloir et donc un facteur de branchement plus grand entre les états. En figure 6.5, la courbe autonome nous indique que la politique observée varie fortement de la politique optimale, les méthodes de prédiction auront donc plus d'apprentissage à faire.

Nous pouvons observer que `LEARN` ne parvient plus à apprendre la politique observée. Ceci est dû au nombre d'erreurs, mais également car contrairement au labyrinthe, les couloirs ne limitent plus les effets de bord de la généralisation de la politique. `DIST`, n'est pas affecté par cette généralisation, mais a les mêmes problèmes de stabilité que dans l'environnement précédent. De même, TAMER&RL reste stable mais ne peut apprendre toutes les préférences.

Environnement ouvert, déterministe et opérateur non-expert

Si l'opérateur hésite, alors les méthodes `DIST` ne parviennent plus non plus à se rapprocher de la politique exécutée, d'après la figure 6.6.

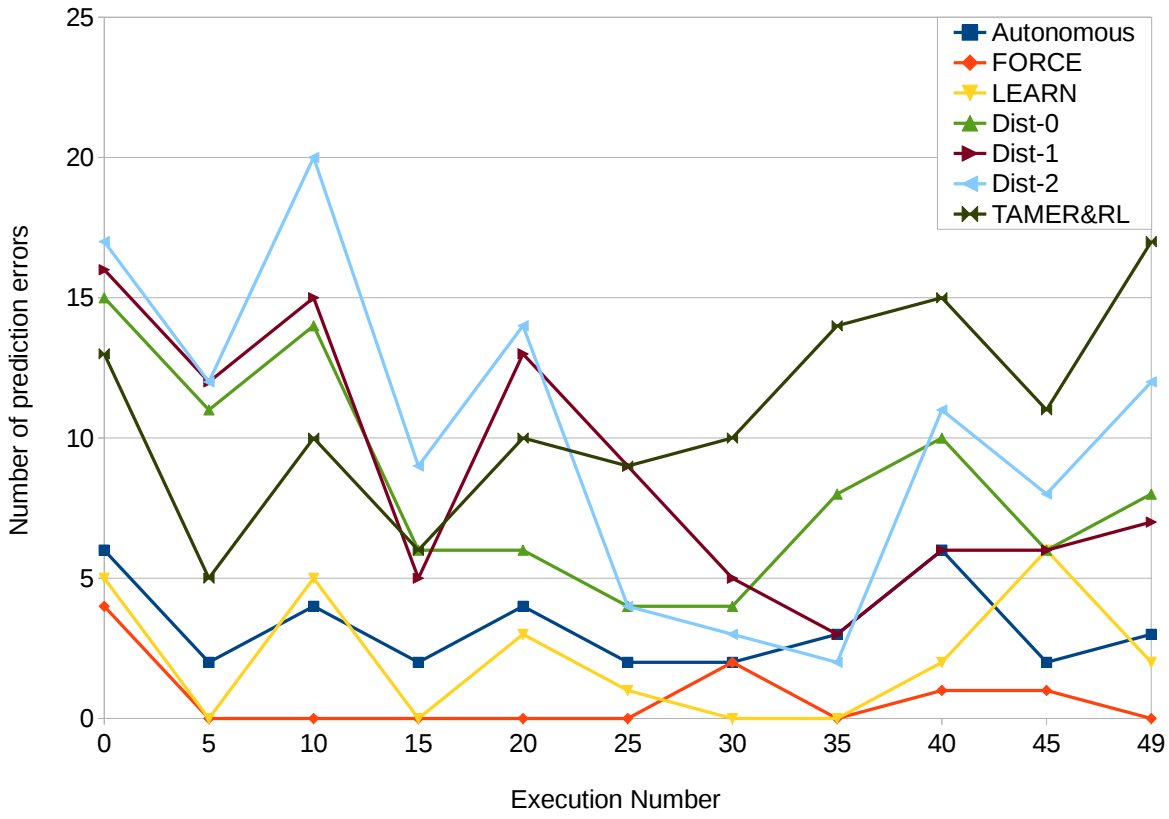


FIGURE 6.4 – Environnement fermé, stochastique et opérateur non expert

Environnement ouvert, stochastique et opérateur expert

La figure 6.7 nous montre que dans le cas ouvert, en environnement stochastique, seul `FORCE` parvient à rester stable et à montrer de bons résultats. Les autres méthodes, quant à elles, peinent au mieux à améliorer légèrement la prédiction, comparé à la politique non-optimale.

6.1.4 Synthèse

Pour résumer les résultats, nous avons présenté le tableau 6.1, présentant pour chaque algorithme son efficacité en fonction de son environnement et de l'expertise de l'opérateur. Ainsi, '++' correspond à de très bon résultat, en opposition à '−'. Globalement, `FORCE` est l'algorithme s'en sortant le mieux, suivi de `LEARN` en environnement fermé, de type labyrinthe, et de près par `DIST` pour un environnement stochastique, en fonction de δ. TAMER&RL lui, bien qu'il semble stable dans un environnement déterministe, ne parvient pas à s'adapter complètement à la politique observée. Dans le cadre stochastique, l'apprentissage est inefficace, pire encore qu'en estimant à partir de la politique optimale.

Néanmoins, les performances de `LEARN` et de `DIST` semblent dépendre de la construction de l'environnement, et `DIST` semble particulièrement instable, c'est pourquoi nous avons envisagé les améliorations décrites en section 4.3.

Dans la section suivante, nous présenterons les expériences visant à évaluer les améliorations apportées en vue de stabiliser l'apprentissage des méthodes de prédiction. Nous verrons également d'autres méthodes que TAMER&RL pour comparer nos algorithmes.

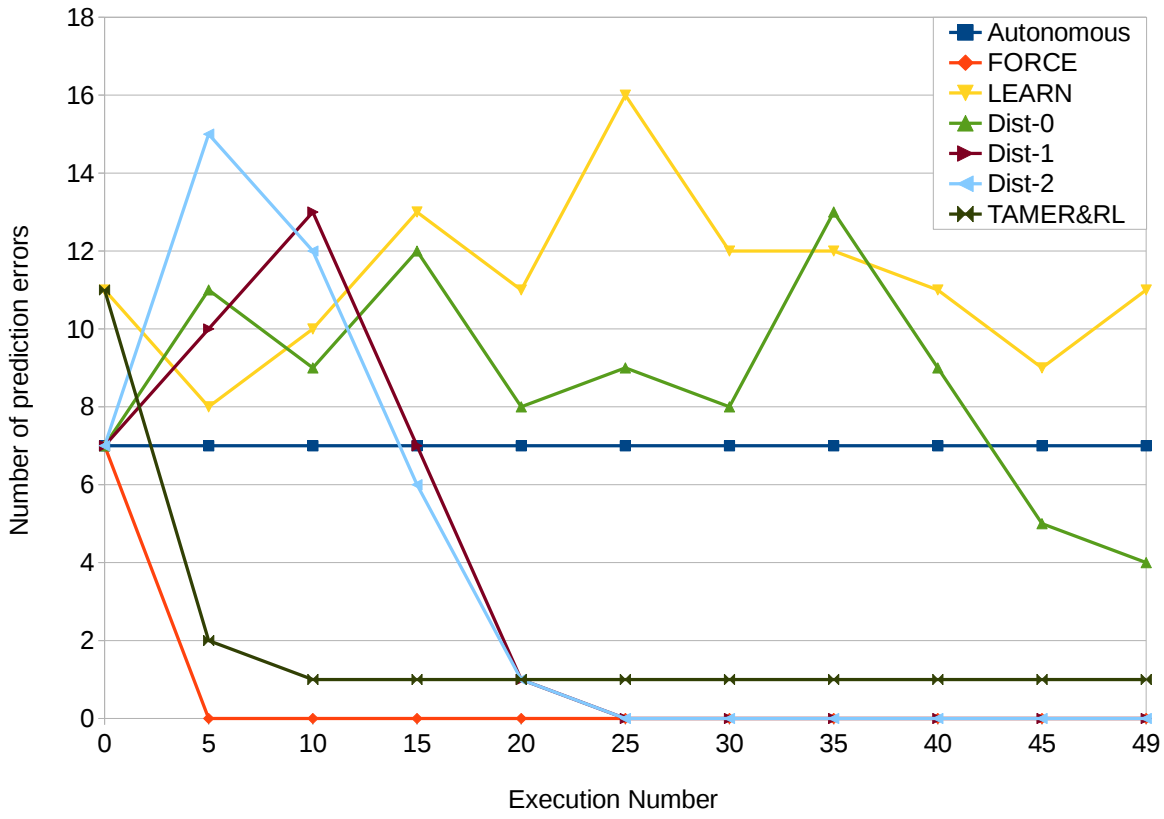


FIGURE 6.5 – Environnement ouvert, déterministe et opérateur expert

6.2 Amélioration de LEARN et de DIST

Cette seconde expérience avait le même but que la précédente, exceptée que beaucoup plus de politiques seraient évaluées. En effet, avec toutes les améliorations possibles de politiques décrites en 4.3, LEARN et DIST ont beaucoup de variantes. Cette expérience a valu une publication à la conférence ICTAI [Lelerre *et al.*, 2017].

Afin de comprendre l'impact de chaque amélioration des méthodes, nous avons fait nos tests en nommant les méthodes de la façon suivante :

- Chaque méthode ajoutée est une dérivée de LEARN ou de DIST, elle contient le nom de la méthode d'origine.
- Une méthode prenant en compte les paramètres des états, c'est à dire les informations fournies par l'opérateur (section 4.3.2), finit par lettre **P**.
- Une méthode combinant variables composant l'espace d'états, et paramètres fournis par l'opérateur contient par la lettre **C**, juste avant le nom de la méthode source.
- Une méthode appliquant la suppression des maximums locaux, comme défini en section 4.3.3, commence par le caractère +.

6.2.1 Scénario d'expériences

Modèle : Comme l'objectif est semblable au précédent, le scénario est proche. Néanmoins, les cartes ont été changées afin de vérifier l'impact des améliorations des politiques.

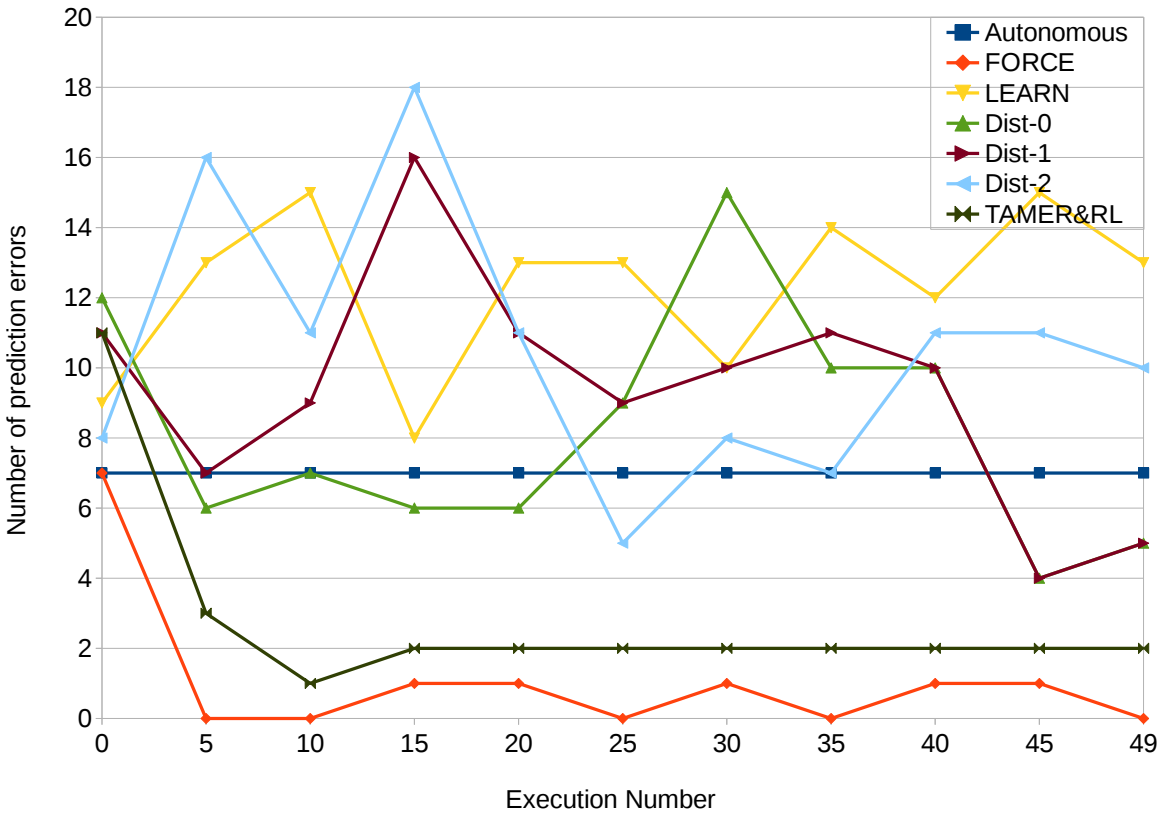


FIGURE 6.6 – Environnement ouvert, déterministe et opérateur non-expert

Concernant l'état de l'agent, nous considérons qu'il n'est plus en mesure de capter la lumière de son environnement, ou d'éclairer la pièce en fonction de l'éclairage. Néanmoins, l'éclairage de l'environnement est considéré comme fourni par l'opérateur en tant que propriété de l'état du robot. L'éclairage ne fait donc plus parti des variables d'état.

Comme l'agent ne peut plus adapter l'éclairage, ses seules actions possibles sont le déplacement dans la carte. De même, la fonction d'utilité ne fait plus référence à l'éclairage. Elle représente donc un coût faible pour chaque action, et une récompense pour atteindre l'objectif.

Comportement simulé : Le comportement de l'opérateur va être simulé de la même façon que dans l'expérience précédente. Néanmoins, les changements de comportement simulant les préférences de l'opérateur seront basées sur les paramètres de l'environnement fournis à l'agent. Par exemple, l'opérateur évitera les états associés à certaines valeurs de paramètres. Les autres influences sur l'environnement, comme la précision de l'opérateur, ou l'hésitation, n'en dépendront pas.

6.2.2 Méthodes de comparaison

Apprentissage par renforcement inverse

Présentation Une première méthode que nous avons envisagé est l'apprentissage par renforcement inverse, qui consiste à retrouver la fonction d'utilité d'un opérateur pilotant un robot. [Ng

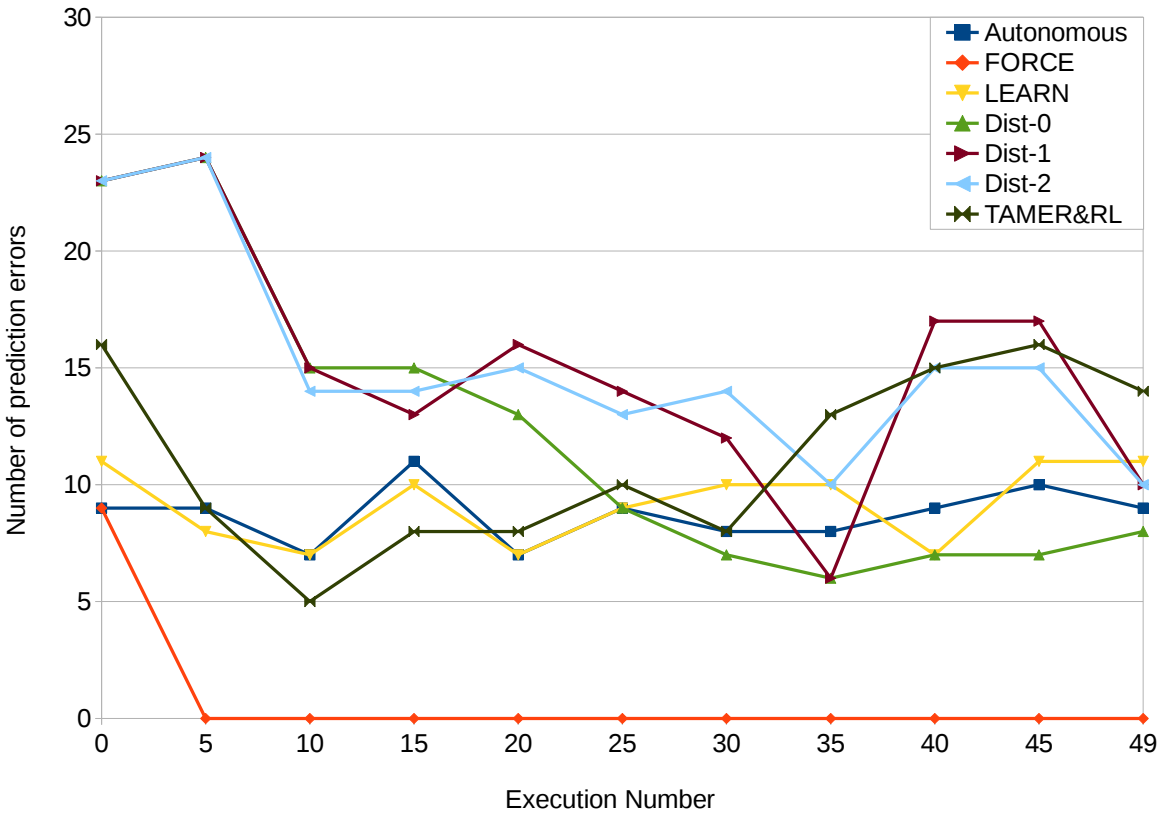


FIGURE 6.7 – Environnement ouvert, stochastique et opérateur expert

et Russell, 2000] propose un algorithme exploitant la programmation linéaire (PL) permettant, à partir d'un modèle MDP sans fonction d'utilité, d'une politique π, de traces d'exécution , ou d'une trajectoire H observée (il s'agit d'un ensemble de couples états/actions ordonnés), de calculer une fonction d'utilité permettant de se rapprocher du comportement correspondant.

Afin de générer cette fonction d'utilité r_{irl}, l'algorithme considère une fonction d'utilité composée. Elle est définie à partir d'une combinaison linéraire $r_{irl}(s,a) = \sum_{i=1}^{n} \alpha_i \phi_i(s,a)$, où $\phi_i(s)$ est la $i$ième composante de la fonction d'utilité, et α_i sa pondération. L'objectif de ce modèle est donc de fixer la valeur des pondérations pour se rapprocher de la politique observée. Pour cela, nous utilisons l'algorithme 15.

Cet algorithme utilise le programme linéaire décrit dans l'équation 6.6, permettant de maximiser, pour chacune des politiques précédemment calculées, la différence entre l'utilité espérée de celle-ci et l'utilité espérée de la politique observée, renforçant ainsi l'impact de la différence entre la politique optimale et les autres. La fonction p est une fonction doublant les nombre négatifs, afin de diminuer les cas où une politique est plus efficace que l'optimale. Ainsi, l'algorithme va générer plusieurs politiques, jusqu'à remplir la condition d'arrêt en ligne 3. Cet algorithme permet d'obtenir une politique dite "satisfaisante". Nous considérons par exemple qu'une politique "satisfaisante" correspond à la trajectoire observée. Néanmoins, il est possible de ne pas trouver de fonction d'utilité r_{irl} permettant de générer une politique adéquate. Il est

Tableau 6.1 – Efficiency of prediction methods, considering the situation

environment	Indoor				Outdoor			
transitions	Det.		Stoc.		Det.		Stoc.	
hesitation	non	oui	non	oui	non	oui	non	oui
`FORCE`	++	++	++	++	++	++	++	++
`LEARN`	++	++	++	+	−	−	−	−
`DIST`	+	+	++	−	++	−	−	−
TAMER&RL	+	+	−	−	+	+	−	−

Algorithme 15 : Apprentissage par renforcement inverse

Données : Un tuple $\langle S,A,T\rangle$, définissant un MDP sans fonction d'utilité;
H trajectoire observée;
Φ ensemble des composantes de la fonction d'utilité;

```
1 α_i ← 0 ∀i ∈ [1..n];
2 k ← 1;
3 tant que ⟨condition⟩ faire
4     k ← k + 1;
5     ∀i ∈ [1..n], α_i mis à jour avec le programme linéaire 2.14;
6     π_k ← politique générée par value iteration (algorithme 2 ), avec la fonction d'utilité
        r_irl;
```

donc également envisageable de limiter le nombre de politique à un nombre fixe.

$$\begin{array}{ll} \text{Maximiser} & \sum_{i=1}^{k} p(V^{\pi^*}(s_0) - V^{\pi^i}(s_0)) \\ \text{tel que} & |\alpha_i| \leq 1 \forall i \in [1..n] \end{array} \tag{6.6}$$

Adaptation Cet algorithme permettrait de calculer une fonction d'utilité de l'opérateur représentant ses préférences personnelles. Il serait alors possible de générer une politique à partir de cette fonction, qui servirait alors d'estimation de la politique de l'opérateur. Néanmoins, cet apprentissage se fait sans à priori sur la fonction d'utilité de l'opérateur, contrairement aux modèles `DIST` et `LEARN`. Pour assurer une équité lors des tests, nous ajoutons aux composantes de l'utilité la fonction d'utilité du modèle autonome, c'est à dire la fonction d'utilité permettant de générer la politique initiale des méthodes `DIST` et `LEARN`. Par contre, la pondération de celle-ci est fixée à 1, pour assurer sa prise en compte.

Approche statistique

Principe Une autre approche envisageable est l'approche statistique. Pour cela, nous nous basons sur une partie des travaux de [Le Hy, 2007], visant à générer, pour un agent dans un jeu vidéo, un comportement proche de celui d'un humain, en observant les autres joueurs.

Soit n_s le nombre de fois où le robot se trouve dans l'état s, et $n_{a,s}$, le nombre de fois où l'on y observe l'action a. [Le Hy, 2007] définit une politique stochastique (ou la probabilité qu'un agent effectue une action dans un état donné) π_s d'un agent à partir de l'équation 6.7.

$$\forall s \in S : \pi_s(s,a) = \frac{n_{a,s} + 1}{n_s + |A|} \tag{6.7}$$

L'idée est donc, dans le cas où l'agent n'a pas atteint l'état s, d'avoir autant de chances pour chacune des actions d'y être effectuées. Dans le cas contraire, les actions les plus observées sont les plus probables d'être ré-observées.

Adaptation La politique actuellement générée par cette méthode est stochastique, c'est à dire qu'elle retourne la probabilité d'observer une action dans un état donné. Hors, dans l'ensemble des politiques que nous avons étudiés juqu'à présent, la politique était déterministe, c'est à dire qu'elle retournait l'action estimée directement. Pour revenir à une politique classique, dite , nous avons adapté l'équation précédente afin d'obtenir celle-ci.

$$\forall s \in S : \pi(s) = \underset{a \in A}{\operatorname{argmax}} \frac{n_{a,s} + 1}{n_s + |A|} \tag{6.8}$$

Ainsi, les actions qui avaient le plus de chances d'êtres observées sont les actions estimées par l'observateur.

6.2.3 Résultats

Nous avons étudié deux cas d'expériences : dans le premier, la mission n'était exécutée qu'une seule fois, afin d'évaluer l'apprentissage sur une itération. En effet, l'expérience précédente a montré que de nombreuses méthodes détérioraient la prédiction avant de l'améliorer. Seulement, nous cherchons à ce que l'algorithme permette d'apprendre rapidement la politique. L'autre cas est, comme pour la première série de tests, l'exécution répétée de la mission, permettant de profiter des informations déjà apprises à chaque répétition de la mission.

Contrairement à l'expérience précédente, les résultats ici ont été moyennés pour lisser les courbes et permettre une meilleure interprétation des résultats.

Pour rappel, les méthodes définies en 4.2 commençant par un '+' correspondent aux méthodes implémentant l'amélioration définie en 4.3.3, celles finissant par P n'apprennent que sur les paramètres des états et celles avec comme préfixe le caractère 'C' apprennent à la fois des variables et des paramètres des états.

Cas 1 : Une seule exécution de la mission

La table 6.2 montre le nombre d'erreurs de prédiction observées au cours d'une mission, sans apprentissage. La méthode d'apprentissage statistique n'est pas présentée dans ces résultats car sans parcours au préalable des états, le choix de politique est aléatoire.

La politique d'apprentissage par renforcement inverse n'est pas toujours performante non plus. Il y a deux raisons à cela. La première est que l'algorithme, en apprenant après chaque déviation, apprend la fonction d'utilité le menant dans l'état atteint après la déviation. En fonction des politiques précédemment générées, il peut arriver que l'agent apprenne à rejoindre cet état, plutôt que rejoindre l'objectif en passant par cet état. La deuxième raison est que les composantes d'états sont définies en fonction des propriétés et des variables des états. Hors, certaines variations de la politique, comme les hésitations, ne dépendent pas forcément de ces informations. L'algorithme essaie donc de trouver une fonction d'utilité telle qu'elle maximise l'utilité de la politique observée en opposition aux politiques précédemment analysées, mais comme il n'y a pas toujours cohérence entre la politique observée, et les fonctions utilisées pour générer la fonction d'utilité, le comportement généré peut parfois devenir imprévisible. Ces observations nous ont emmenés à penser que cette solution n'est pas adapté à notre problème.

Tableau 6.2 – Résultats pour une mission exécutée une seule fois

	Indoor			Outdoor		
	Deterministic	Stochastic		Deterministic	Stochastic	
hesitation	30.00%	0.00%	30.00%	30.00%	0.00%	30.00%
+CDIST	6.38	9.01	8.20	5.03	6.85	6.21
+CLEARN	4.68	4.35	4.88	4.20	5.94	5.69
+DIST	6.38	8.85	8.10	5.03	6.85	6.21
+DISTP	6.38	8.23	7.61	7.03	7.05	6.31
+LEARN	5.47	5.88	5.95	4.46	6.05	5.91
+LEARNP	4.37	4.51	4.67	4.95	6.49	6.83
Autonomous	7.38	8.24	7.69	8.46	7.15	6.49
CDIST	6.38	12.92	12.11	6.03	7.03	6.41
CLEARN	10.37	19.46	18.10	5.52	7.90	7.97
DIST	6.38	13.39	11.80	6.03	8.78	7.92
DISTP	7.07	12.97	11.72	7.03	8.21	7.41
FORCE	4.00	4.21	4.35	6.74	6.38	5.83
IRL	9.01	15.46	15.53	3.97	11.82	11.87
LEARN	8.06	13.80	11.98	5.72	6.84	7.06
LEARNP	6.70	6.58	6.76	4.61	6.26	6.55

Concernant les méthodes de prédictions définies en 4.2, nous observons que sans améliorations, celles-ci ont tendance à détériorer la prédiction de la politique dès l'exécution, excepté pour `FORCE`. Néanmoins, apprendre sur les paramètres des états plutôt que sur les variables, permet d'améliorer de manière significative. Ceci s'explique du fait que l'observateur apprend sur les paramètres jugés importants par l'opérateur. Combiner l'apprentissage sur les variables et les états semble par contre détériorer les performances. Ceci est dû au fait qu'il lui faut plus de temps pour généraliser, car il apprend sur un plus grand domaine. Néanmoins, l'amélioration décrite en 4.3.3 permet de stabiliser l'algorithme en évitant les excès d'apprentissage. Au final, en utilisant cette dernière amélioration et en prenant en compte les paramètres, `LEARN` et `DIST` parviennent à être plus efficace que `FORCE` sur un espace ouvert, grâce à la généralisation de l'apprentissage.

Cas 2 : Plusieurs itérations de la mission

Environnement fermé, déterministe, opérateur non expert

La figure 6.8 nous montre que `FORCE` est toujours très rapide comparé au autres méthodes. Néanmoins, la méthode statistique semble plus efficace sur le long terme. En effet, là où `FORCE` enregistre la dernière action exécutée à chaque état, la méthode statistique enregistre l'action la plus exécutée, réduisant le nombre d'erreurs. Elle met en revanche plus de temps à apprendre.

`DIST` semble un peu plus stable, mais plus efficace dans sa deuxième version que lors des tests précédents, mais est plus lente que les autres méthodes, ce qui s'explique par l'apprentissage local des préférences.

La suppression des maximum locaux apporte une meilleure stabilité dans l'apprentissage, et permet à `CLEARN` d'apprendre plus rapidement. Il améliore également les performances sur toutes les méthodes dérivées de `DIST`. `+CDIST` parvient même à être plus performant que `FORCE` sur le long terme. Néanmoins, concernant `DIST`, comme celui-ci ne généralise déjà pas beaucoup,

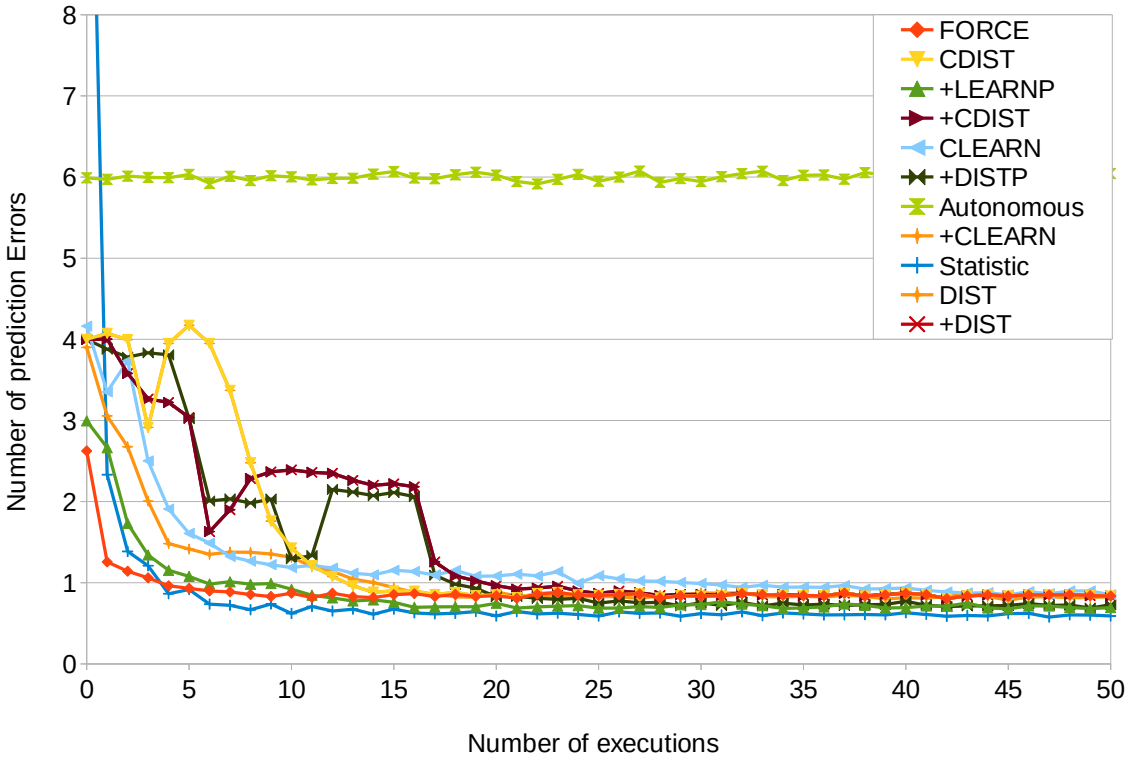


FIGURE 6.8 – Environnement fermé, déterministe, opérateur non expert

la vitesse d'apprentissage est encore diminuée en appliquant cette méthode.

Environnement fermé, stochastique, opérateur expert

La figure 6.9 Montrent que les méthodes `FORCE` et statistiques sont toutes deux plus rapides que les autres. Néanmoins, après plusieurs exécutions de la missions, d'autres méthodes deviennent plus efficace, comme `+DIST`. La plus part des autres méthodes parviennent à avoir une prédiction aussi efficace que `FORCE` ou la méthode statistique, mais il leur faut un peu plus de temps, car un peu moins stables.

Environnement fermé, stochastique, opérateur non-expert

La figure 6.10 montre une fois de plus que lorsque l'opérateur hésite, la méthode statistique est plus efficace que les autres. Néanmoins, `+learnp` apprends rapidement et se rapproche de cette méthode en une quinzaine d'itérations. `+distp` présente des résultats proches sur le long terme, mais est plus instable sur le court terme. En effet, comme déjà dit plus tôt, les méthodes dérivées de `DIST` ne généralisent pas, et mettent plus de temps à apprendre. Cet effet est accentué par la suppression des maximum locaux. En effet, bien qu'il diminue la génération d'erreurs d'estimations, que l'on peut voir en comparant avec `CLEARN`, elle met plus de temps à se stabiliser. Sur le plus long terme, là où la plus part des méthodes semblent se stabiliser, la méthode statistique continue d'apprendre.

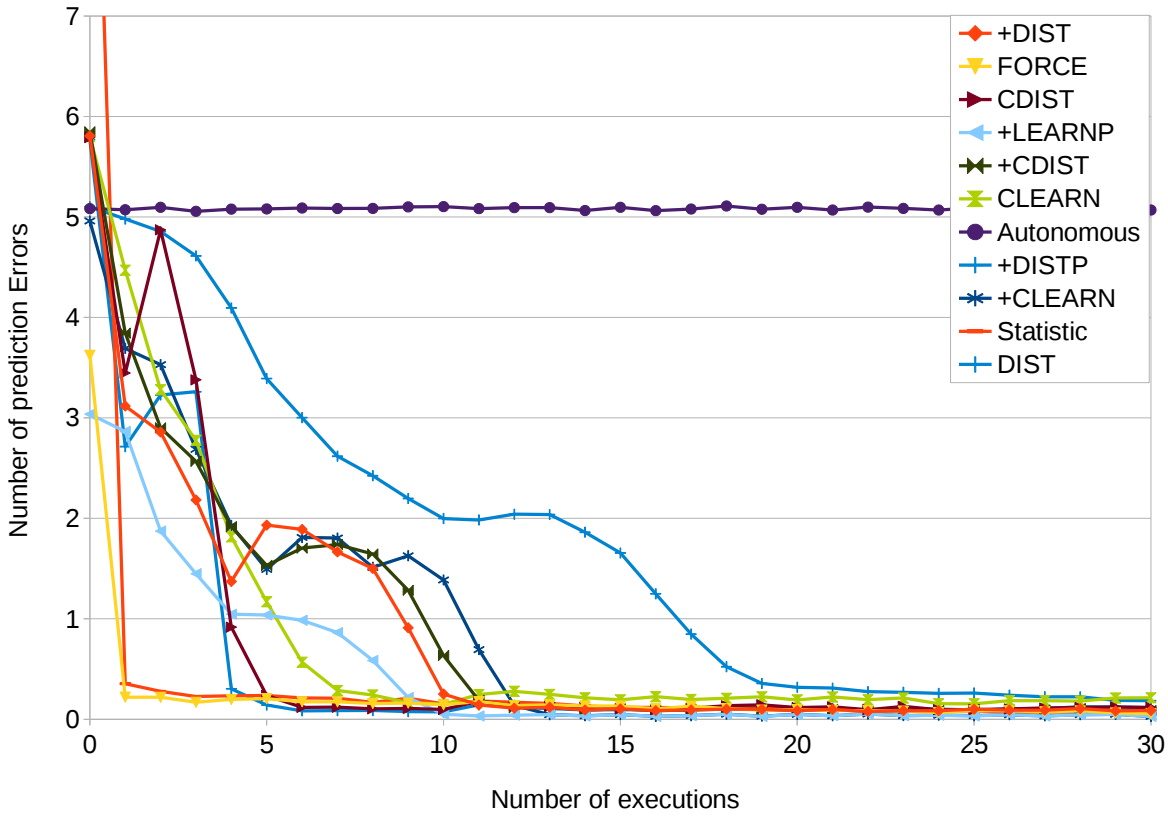


FIGURE 6.9 – Environnement fermé, stochastique, opérateur expert

Environnement déterministe, ouvert, opérateur non-expert

La figure 6.11 comfirme les résultats précédents concernant `+CDIST` et `+DISTP`. Tout comme un environnement fermé, avec plus d'obstacles, la méthode statistique génère une politique plus proche de celle observée.

`CLEARN`, avec ou sans l'amélioration décrite en section 4.3.3, visant à supprimer les fortes récompenses parasites, semble être plus lente que les autres méthodes. `LEARN` était déjà moins efficace dans notre précédente expérience dans un environnement ouvert, ce qui peut partiellement expliquer ce résultat. Une autre explication possible est le fait qu'il y ait plus de variables sur lesquelles apprendre, rendant l'équilibrage de l'apprentissage des préférences sur l'ensemble de ces facteurs plus long.

Malgré tout, `DIST` et `LEARN` sont plus stables avec les améliorations apportées lors de ces expériences. Par exemple, `DIST` et `CLEARN` génèrent une estimation de la politique aussi efficace que `FORCE` en moins de 10 itérations de la mission.

Environnement stochastique, ouvert, opérateur expert Tout comme en environnement stochastique, nous pouvons voir à partir de la figure 6.12 que la méthode `FORCE` est plus efficace que la méthode statistique. Par contre, les méthodes `DIST` et `CLEARN` parviennent également à estimer la politique exécutée plus efficacement que la méthode statistique aux premières itérations de la mission, avant d'être équivalentes.

La réduction des états avec fortes récompense a néanmoins tendance à fortement ralentir les

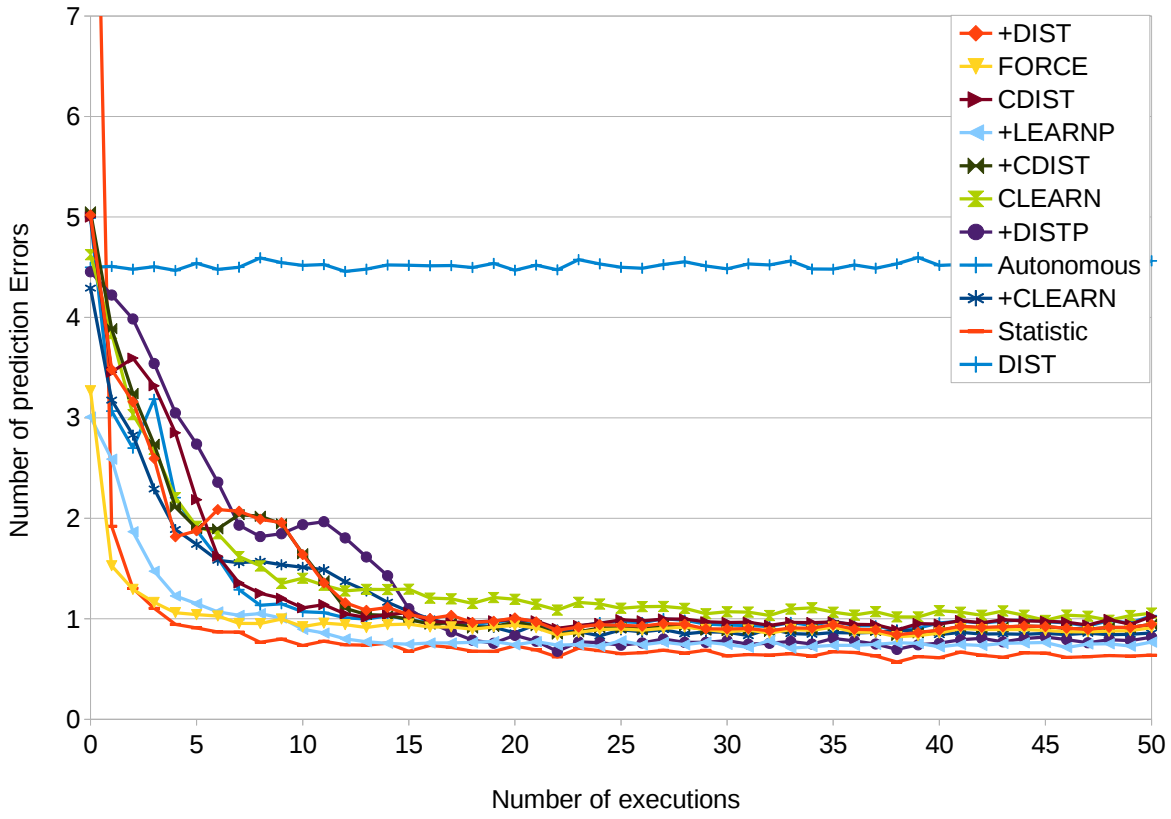


FIGURE 6.10 – Environnement fermé, stochastique, opérateur non-expert

méthodes dérivées de DIST. La méthode +DISTP semble même finir par apprendre des erreurs sur un très grand nombre d'itérations de mission. Ceci viendrait d'une politique non optimale, qui ne serait pas due à des préférences particulières de l'opérateur sur les paramètres des états, que la méthode tenterait d'assimiler à une préférence sur des paramètres.

Environnement stochastique ouvert, opérateur non-expert

La figure 6.13 nous montre que les hésitations augmentent les erreurs générées par la correction des états à fortes récompense sur des méthodes raisonnant sur les paramètres des états uniquement. En effet, +DISTP, +LEARNP ne parviennent pas à apprendre efficacement. De même, +DIST et +CDIST mettent plus de temps à apprendre.

Les méthode FORCE et statistique présentent de très bons résultats tout au long de l'exécution, bien que la méthode statistique se démarque des autres méthodes.

DIST et CLEARN présentent de très bons résultats dans les premières itérations. Néanmoins, la méthode statistique finit par être plus efficace au bout d'une quinzaine d'itérations de la mission, sans pour autant être plus efficace que la méthode FORCE.

Les autres méthodes apprennent efficacement, et finissent par se stabiliser après environs une dizaine d'itérations. Elles parviennent tout de même à faire en moyenne moins d'une erreur par itération de la mission, ce qui est tout à fait acceptable.

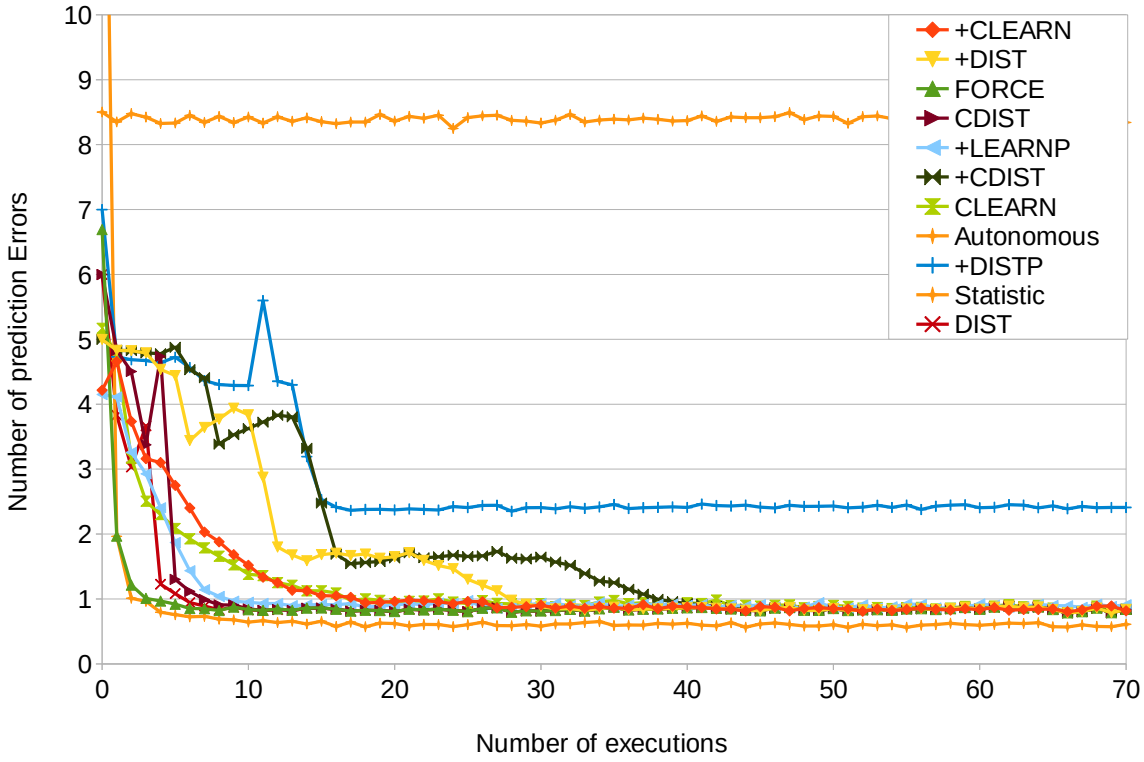


FIGURE 6.11 – Environnement déterministe, ouvert, opérateur non-expert

6.2.4 Conclusion

Cette dernière série d'expériences a montré une nette amélioration des méthodes de prédictions présentées précédemment. Elles mettent en évidence une efficacité certaines des améliorations des méthodes dans le cas où la mission est effectuée une seule fois, mais également lorsque celle-ci est répétée dans le cadre d'une patrouille. Elle révèle que la méthode `FORCE`, bien que battue dans certaines situations et face à certaines améliorations, reste un bon compromis puisqu'elle présente de bons résultats dans tous les cas.

Néanmoins, dans le cadre de missions répétées, comme lors de patrouille, nous avons mis en évidence certaines propriétés des différentes méthodes et améliorations. `DIST`, par exemple, semble plus lente à apprendre, due au manque de généralisation. De même, `LEARN` généralise trop, si on ne prend pas en compte les paramètres définis par l'opérateur. Au final, sur le long terme, une méthode statistique sera généralement plus efficace.

Prendre en compte les paramètres seulement peut permettre d'améliorer l'apprentissage. Néanmoins, combinée à la suppression des préférences positives, cette amélioration peut présenter des résultats instables, en fonction des raisons qui poussent l'opérateur à choisir une action non optimale par rapport à une politique optimale. En combinant ces paramètres avec les variables des états, l'apprentissage est plus long, car la généralisation se fait sur plus de facteur, mais permet d'être plus précise.

La suppression des objectifs parasites permet d'améliorer la stabilité de l'algorithme en évitant que la méthode prédise un objectif différent de l'original. Néanmoins, l'apprentissage est

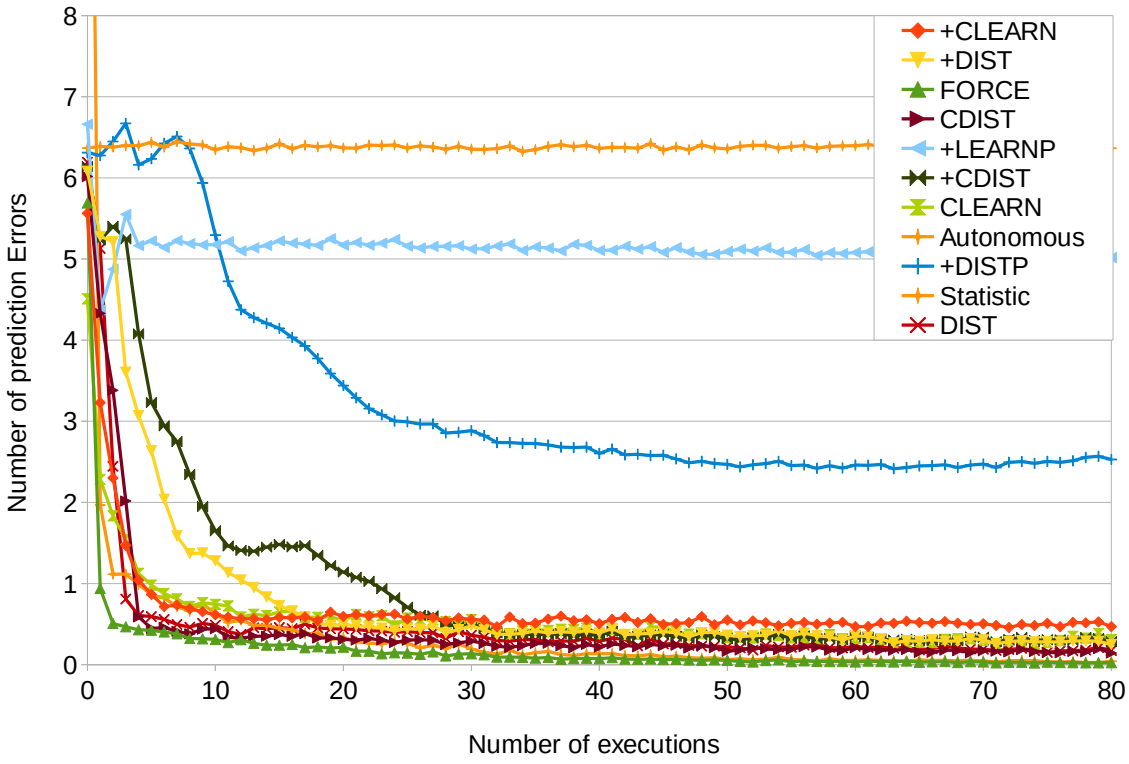


FIGURE 6.12 – Environnement stochastique, ouvert, opérateur expert

ralenti, car les méthodes n'apprennent plus les politiques préférées, mais seulement les politiques évitées. Elles apprennent donc par élimination des actions évitées plutôt que par sélection des actions observées.

Nous avons pu noter une amélioration générale des résultats obtenus, néanmoins nous avons noté que les raisons de l'opérateur le poussant à dévier de la trajectoire optimale influait sur la capacité des algorithmes à apprendre plus rapidement.

Il reste néanmoins à se demander si la génération des hésitations n'était pas un peu trop simple pour représenter véritablement le type d'hésitations dont les méthodes pourraient être confrontés. En effet, l'opérateur évolue, et au fil des missions ses hésitations pourraient s'atténuer. Au contraire, dans nos expériences, non seulement les hésitations étaient définis sur des états, mais on ne considérait pas qu'elles pouvaient diminuer en même temps que l'opérateur s'adaptait à la mission.

6.3 Expérimentations avec le robot NERVA

Suite à ces expériences, nous avons tenté de reproduire ces résultats sur un NERVA, robot développé par NEXTER ROBOTICS, et utilisés pour le projet GARDES.

Notre objectif était de permettre, dans un premier temps de définir un endroit à atteindre pour le robot dans un bâtiment, dont la carte est connue. Par la suite, un opérateur devait piloter ce robot, en le faisant choisir un chemin autre qu'une ligne droite pour atteindre ce point. La

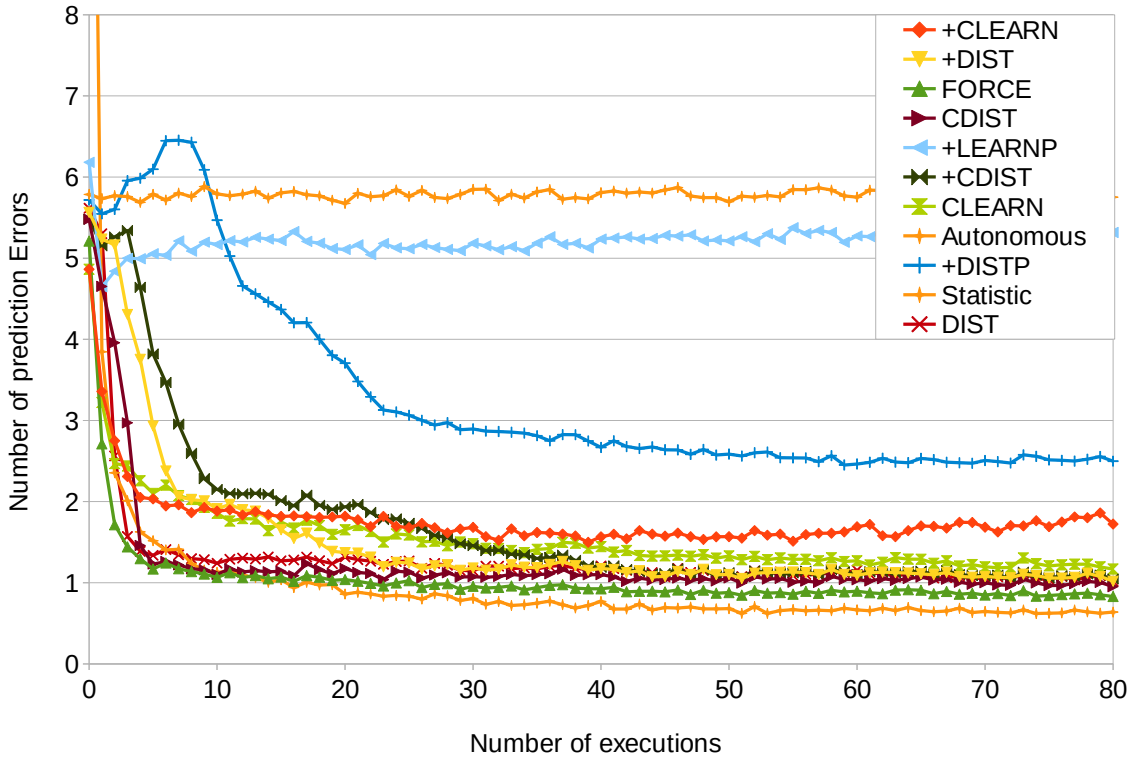


FIGURE 6.13 – Environnement stochastique ouvert, opérateur non-expert

figure 6.14 présente le scénario, où un opérateur utilise un ordinateur pour piloter le robot, tandis que le robot envoie sa position et le retour de ses capteurs (caméra, GPS, carte...) vers l'ordinateur, qui retourne à l'opérateur ces informations, ainsi qu'une estimation de la politique suivie par l'opérateur.

Les robots étant équipés de lazer SLAM, la précision de sa localisation dans l'environnement permet de considérer l'état du robot observable. Dans le cadre d'une mission de navigation avec environnement connu, ces conditions nous permettent de considérer l'état de l'environnement comme étant totalement observable.

Pour réaliser cette expérience, il nous a fallu dans un premier temps mettre en place une interface, permettant de piloter le robot, et de transmettre les propriétés des états. Enfin, il fallait que l'interface permette de visualiser la trajectoire prédite par le robot.

6.3.1 Interfaçage

Les besoins de l'opérateur quant à la télé-opération du robot étaient déjà disponibles. En effet, NEXTER-ROBOTICS a mis à disposition une interface, présenté en figure 6.15, permettant de piloter le robot à partir de CHECKPOINT, ou en pointant une direction via les mouvements de la souris, tout en récupérant le flux vidéo des caméras du robot.

Pour le reste, il a néanmoins fallu redéfinir une interface. D'autres membres du laboratoire on développé, pour le projet GARDES une interface réseau en JAVA permettant de communiquer avec un NERVA, et ainsi de récupérer la carte du robot ainsi que sa position. Néanmoins, il ne

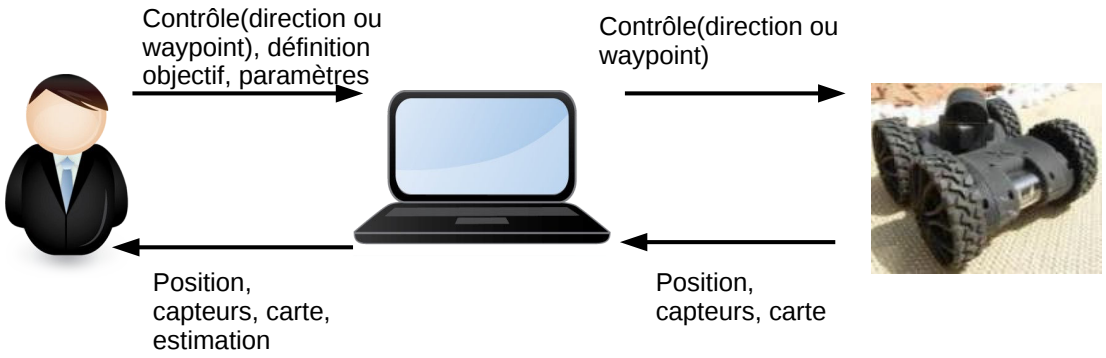


FIGURE 6.14 – Présentation du scénario

nous est pas possible de récupérer directement l'action exécutée par le robot. Nous observons donc l'action effectuée en interprétant le changement d'état du robot.

A partir de ces informations, nous avons défini une nouvelle interface permettant d'afficher la carte du robot, et de dessiner dessus afin de définir les propriétés des états de la carte, ou de définir un objectif de mission pour en déduire une politique optimale initiale. D'autre part l'interface permet de définir des méthodes de prédiction. On en retrouve une représentation en figure 6.17, avec en rouge un exemple de prédiction de politique pour les dix prochaines actions, en vert l'objectif fixé par l'opérateur et en bleu une indication apportée par l'opérateur. Le panneau latéral permet de choisir les politiques d'estimation à utiliser, et à renseigner les paramètres des états à renseigner.

Avant, ou pendant la mission, l'opérateur définit les propriétés des états et un objectif et les méthodes avec lesquelles il souhaite prédire ses futures actions. L'opérateur pilote ensuite le robot avec l'interface de pilotage. Pendant ce temps, l'interface affiche en temps réel la position du robot, et les prochains états atteints à un horizon fini. Lorsque l'opérateur dévie de la politique estimée, cette transition est placée en buffer des méthodes de prédictions tant qu'elles sont en train de mettre à jour les politique estimées dans un second thread. La trace affichée est ainsi la dernière politique estimée finie d'être générée.

6.3.2 Observation

Nous avons pu observer au cours des exécutions que les méthodes permettent de mettre à jour la politique en présentant une trajectoire proche de celle observée. Néanmoins, il existe des cas où certaines politiques sont moins efficace que d'autres. Par exemple, si le robot est loin de la position incitant l'opérateur à dévier de la politique optimale, alors celui-ci n'est pas en mesure de définir avec `DIST` ou `LEARN` pourquoi cette déviation a eu lieu. Ce problème est surtout présent lorsqu'un état doit être évité, puisque l'opérateur rejoint les états souhaités. Néanmoins, apprendre sur les variables des états atténue légèrement ce facteur, dans le sens où elle permet de définir les états menant vers les états problématiques.

`FORCE` est une méthode qui est efficace, mais dont la prédiction de la première itération ne donne que peu d'informations sur la trajectoire envisagée. Elle apporte surtout pour une seconde transition.

Nous avons pu également remarquer que certaines déviations pouvaient être parasitaires. En effet, le contrôle du robot se faisant en déplaçant la souris, les mouvements n'étaient pas toujours très précis, impliquant des déviations qui n'étaient pas voulues. Ce problème peut néanmoins

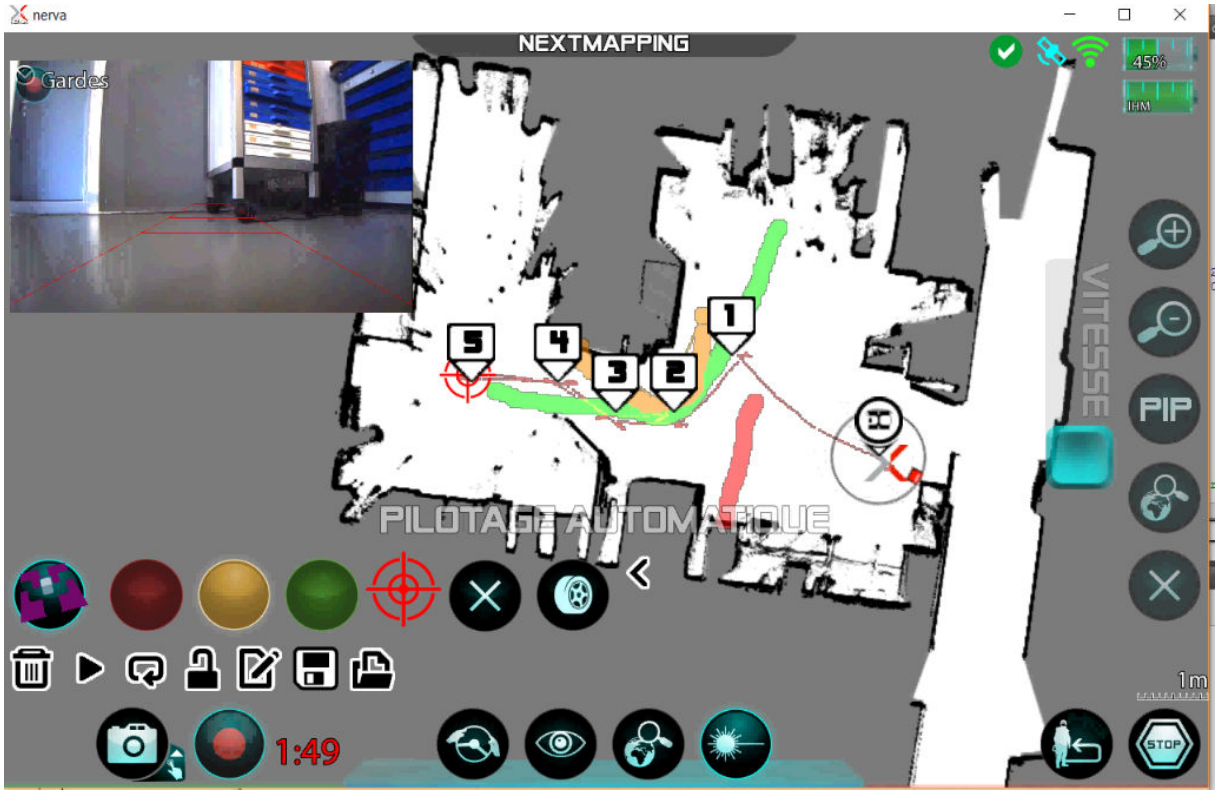


FIGURE 6.15 – Interface Nexter

être atténué par la prise en compte des paramètres des états.

6.4 Conclusion

Nous avons présenté, au cours de ce chapitre, plusieurs expériences visant à tester l'efficacité et la stabilité des algorithmes d'estimation de politique présentés au cours de cette thèse. Bien que dans un premier temps elles aient pu présenter des problèmes de stabilités, nous avons proposé plusieurs solutions augmentant à la fois l'efficacité de la stabilité de nos méthodes, comme la suppression des objectifs parasites ou la prise en compte des paramètres des états, définis par l'opérateur.

Nous avons en outre déterminé que `FORCE` était efficace dans tous les cas, mais pouvait être moins efficace que d'autres, notamment dans des environnements stochastiques. `DIST` est plus lent que `LEARN`, mais permet parfois de générer une meilleure estimation que `FORCE`. Néanmoins, `LEARN` et `DIST` permettent une meilleure généralisation de la politique que `FORCE` si la mission ne doit être exécutée qu'une seule fois.

La prise en compte des paramètres permet d'améliorer la stabilité et l'efficacité des méthodes, met ralentit l'apprentissage si celle-ci est combinée avec les variables des états. La suppression des objectifs globaux apporte une meilleure stabilité, mais ralentit l'apprentissage de manière générale. Ainsi, combinée à `DIST`, l'apprentissage devient trop lent pour être vraiment intéressant.

Enfin, ces méthodes sont battues sur le long terme par une méthode statistique. En fonction de la situation, et notamment de la stochasticité de l'environnement, il peut néanmoins falloir un certain nombre d'itérations de la mission avant que l'estimation générée par les statistiques

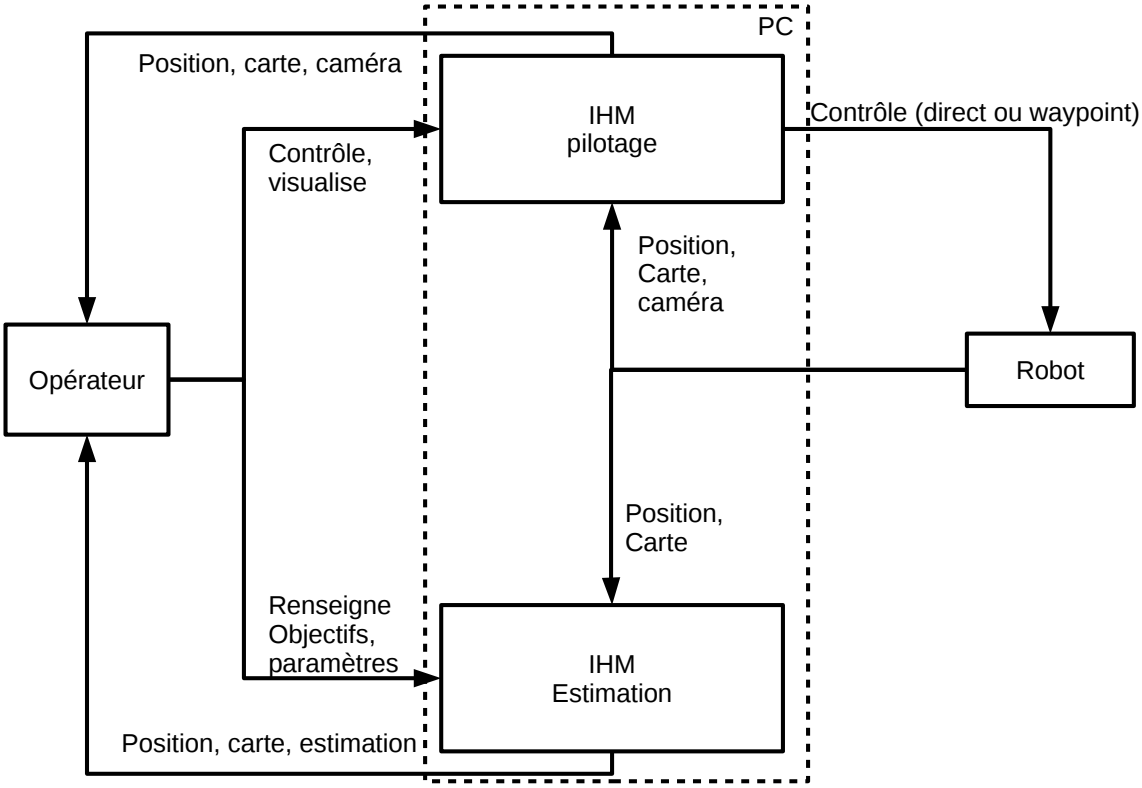


FIGURE 6.16 – Interactions entre robot, interface, et opérateur

ne soit meilleure que celle générée par nos méthodes.

Après ces expériences, plusieurs questions se posent. D'une part, nos hésitations ont été définies en fonction de l'expertise d'un opérateur, sur une liste définie d'état. Néanmoins, l'opérateur n'hésitera pas forcément systématiquement aux mêmes endroits, et celui-ci est amené à évoluer au fur et à mesure qu'il connait la mission. Il serait intéressant de voir l'évolution des résultats sur un modèle d'un opérateur plus réaliste.

De même, pour le calcul de la politique estimée, nous avons modélisé les préférences de l'opérateur, mais pas les hésitations de l'opérateur. il pourrait être intéressant d'apprendre quand l'opérateur est susceptible d'hésiter, afin d'éviter de généraliser une hésitation ou de prévoir quand les actions de l'agent télé-opéré seront moins prévisibles.

Nous avons pu voir au cours de nos expériences qu'il est parfois difficile de déterminer l'origine de la déviation de l'opérateur par rapport à la politique optimale, et donc quelle préférence est concernée. Il pourrait être intéressant, afin d'étudier cette lacune, d'étudier des concepts tels que l'affordance, le focus et le nimbus, présentés par [Zieba *et al.*, 2010], utilisés pour évaluer la capacité d'un agent à détecter ce qu'il va l'intéresser.

D'autre part, nous pouvons noter que nos méthodes visent à établir les préférences d'un opérateur. Hors, l'opérateur peut ne pas être le même tout au long de la mission. Il serait alors intéressant de définir un profil d'un opérateur, et de voir si les données récupérées au cours d'une mission sont exploitables pour une autre mission. Par exemple, si un opérateur préfère passer par des coins éclairés, cette connaissance est peut-être réutilisable pour une autre mission, avec le même opérateur.

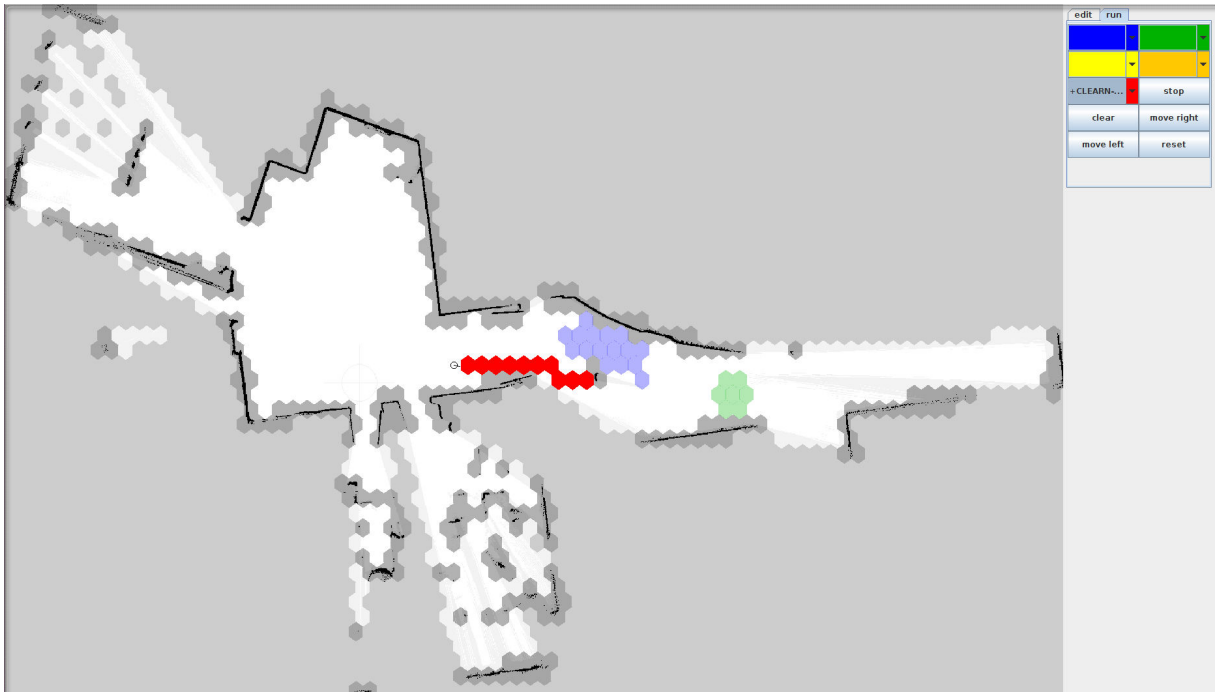


FIGURE 6.17 – Illustration de l'interface d'estimation

Enfin, nous avons évalué l'estimation des politiques générés par nos méthodes. Il reste à évaluer l'efficacité d'un agent se coordonnant avec des agents télé-opérés dont la politique est estimée à partir de ces méthodes.

Chapitre 7

Résultat expérimentaux sur l'autonomie ajustable

Sommaire

L'objectif de cette expérience est de démontrer l'efficacité des Restricted-MDP au chapitre 5. Pour cela, il nous faut dans un premier temps préparer un scénario d'études. Dans un second temps, il faut définir sur quels critères évaluer la politique. Nous présenterons ensuite nos expérimentations sur un agent devant respecter des contraintes fixées par l'opérateur, et nous présenterons les résultats des modèles proposés au cours de ces expériences. Nous conclurons enfin à partir de ces résultats sur la fiabilité et l'efficacité de ces modèles.

7.1 Préparation

7.1.1 Scénario

Dans le scénario étudié, un opérateur supervise un robot, qui a pour mission d'atteindre un point dans un bâtiment. Néanmoins, l'opérateur définit des zones de restrictions pour le robot. L'agent doit alors interagir avec l'opérateur pour pouvoir la traverser. Soit l'opérateur répond favorablement à l'interaction, et l'agent peut traverser, soit il doit trouver une alternative. Si l'agent se arrive par erreur dans la zone restreinte, il a une politique par défaut à appliquer, le menant vers sa position à l'état initial du système, et ce jusqu'à la sortie de la zone restreinte. Ceci a pour intérêt de l'empêcher de traverser la zone via la politique par défaut, sans pour autant le bloquer totalement dans la zone restreinte.

L'agent possède des estimations de réponses de l'opérateur en fonction du niveau de restriction à respecter. Ces connaissances sont définies par l'opérateur. L'agent estime également, lorsque l'opérateur pilote, que celui-ci suit une politique optimale, ne prenant pas en compte les restrictions qu'il a défini.

Nous étudierons plusieurs types d'opérateurs :

— Un opérateur peu disponible (Busy), ne répondant favorablement qu'à 10% des requêtes de l'agent.
— Un opérateur assez disponible (Half), répondant favorablement à 50% du temps aux requêtes de l'agent.
— Un opérateur attentif (Ready), répondant régulièrement aux demandes de l'agent (à 90% du temps).

Ces probabilités ne sont valables que tant que les agents sont au niveau d'attention `READY`. S'ils sont occupés (`BUSY`), alors ils ont 99% de chances de rester occupés, et 1% de chances de redevenir disponible. Cette probabilité sert à modéliser le temps nécessaire pour finir de superviser un autre agent.

Ceci permettra de voir l'impact de ce type d'opérateurs sur la capacité de l'agent à se débrouiller autrement qu'en lui demandant de l'aide.

Afin d'étudier l'impact des restrictions, nous avons étudié différentes situations associé à la zone restreinte. Dans un premier temps, nous avons fait varier le niveau de restriction de la zone, étant donné que la nature des interactions avec l'opérateur est différente en fonction des niveaux.

Ensuite, nous avons cherché différents impacts de restrictions. En fonction des zones de restrictions, l'agent peut avoir différentes réactions, au vu du nombre d'alternatives à sa disposition. Nous étudierons l'impact des alternatives disponibles pour l'agent à travers trois cas, présentés en figure 7.1 :

— L'agent a une alternative peu coûteuse (cas nommé Closed_Alternative, en gris).
— L'agent a une alternative coûteuse (cas nommé Far_Alternative, en gris et jaune en même temps).
— L'agent n'a pas d'alternative (cas nommée No_Alternative, en rouge).

Les cases bleues et vertes sont respectivement la position de l'agent à l'état initial et à l'objectif.

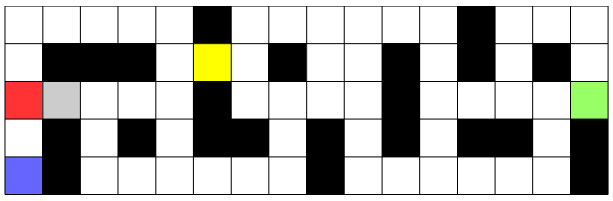


FIGURE 7.1 – Présentation des positions des restrictions associées aux alternatives possibles.

Pour s'assurer qu'il n'y a pas de combinaisons de restrictions venant perturber l'analyse de chaque cas, nous n'aurons qu'une restriction par scénario exécuté.

L'agent peut atteindre des situations où il n'est pas en mesure d'atteindre l'objectif. Pour éviter que le programme ne puisse terminer, nous arrêtons la simulation après 200 actions effectuées, où nous considérons que l'agent ne pourra plus atteindre l'objectif. Nous moyennons nos résultats sur 100 exécutions.

7.1.2 Critères d'évaluation

Pour évaluer l'efficacité des Restricted-MDP présentés au chapitre 5, nous avons dans un premier temps défini les critères permettant d'évaluer l'efficacité de l'autonomie ajustable.

[Zilberstein, 2015] évoque plusieurs solutions, comme le fanout, qui est une mesure, basée sur le rapport entre le temps de négligence du robot et le temps d'interaction avec le robot, permettant d'estimer le nombre de robot supervisable par un opérateur. Dans notre cas, il nous est difficile d'établir le fanout étant donné que les restrictions imposées au robot sont fortes, c'est à dire que l'agent ne peut aller à l'encontre d'une restriction. En d'autres termes, si la restriction sur le robot est trop importante, ce n'est pas qu'il lui est difficile d'accomplir la mission, mais presque impossible. Il peut également arriver que l'agent considère que demander l'intervention de l'opérateur, nécessaire pour l'accomplissement de la mission, serait plus coûteuse que l'échec de la mission. Nous préférons donc évaluer séparément le temps nécessaire pour accomplir la mission, en nombre d'actions, et le nombre de demandes d'interactions de la part de l'agent. Ce nombre correspond au temps où l'agent demande un niveau d'attention de l'opérateur supérieur à `READY`, additionné au nombre d'autorisations demandées.

Un autre critère d'évaluation pour les systèmes semi-autonomes est le nombre de fois où l'agent est en situation de *dead-end*, c'est à dire totalement bloqué dans le système. Il peut être difficile d'interpréter dans un MDP lorsque l'agent est bloqué. En effet, il faut s'assurer que l'agent n'alterne pas entre deux positions en espérant, avec une transition stochastique, de basculer dans un état inattendu. L'agent peut également tenter d'entrer dans une zone restreinte en espérant que l'opérateur change son niveau d'attention, quitte à suivre la politique par défaut associé à la zone restreinte, puis y retourner après. Pour interpréter ces différents cas comme des situations de *dead-end*, nous considérons qu'un agent qui atteint un état qu'il a déjà atteint plus tôt est dans un *dead-end*, et ce jusqu'au prochain nouvel état atteint. Néanmoins, cela peut avoir une légère incidence sur le nombre de *dead-end* enregistrés en environnement stochastique, puisque l'agent peut revenir dans un état déjà atteint de manière imprévue.

Afin d'estimer l'apport des politiques introduites, nous comparons nos résultats avec une politique optimale autonome restreinte, sans capacité d'interaction avec l'opérateur.

7.2 Résultats

Les expérimentations sont encore à leurs débuts, et les résultats présents ici sont susceptibles d'être améliorés à l'avenir.

Au cours de nos expérimentations, nous avons évalué deux cas : le cas où l'environnement est déterministe, et l'autre stochastique.

Les différents graphiques présentent les moyennes sur 1000 itérations, pour chaque politique et chaque situation (type d'opérateur, type de restriction).

7.2.1 Environnement déterministe

La figure 7.2 présente le nombre moyen de deadends rencontrés par l'agent dans le cadre déterministe. La figure 7.3 montre quant à elle le nombre d'interactions initiées par l'agent. La figure 7.4 montre en fin la durée de la mission en nombre d'actions.

Nous pouvons observer que l'agent ne rencontre presque jamais de dead-end si des alternatives sont disponibles. Les rares de blocage cas sont ceux où l'agent, après avoir reçu un refus de la part de l'opérateur, répète sa demande, ou suit la politique par défaut de la restriction qu'il a atteint, le ramenant en arrière. Dans le cas où l'opérateur est très occupé, l'agent ne se donne pas la peine de lui demander d'intervenir, et choisit une alternative dans la mesure du possible, tel qu'on peut le voir par exemple dans le cas de la télé-opération, où l'agent ne demande jamais à un opérateur régulièrement occupé d'intervenir.

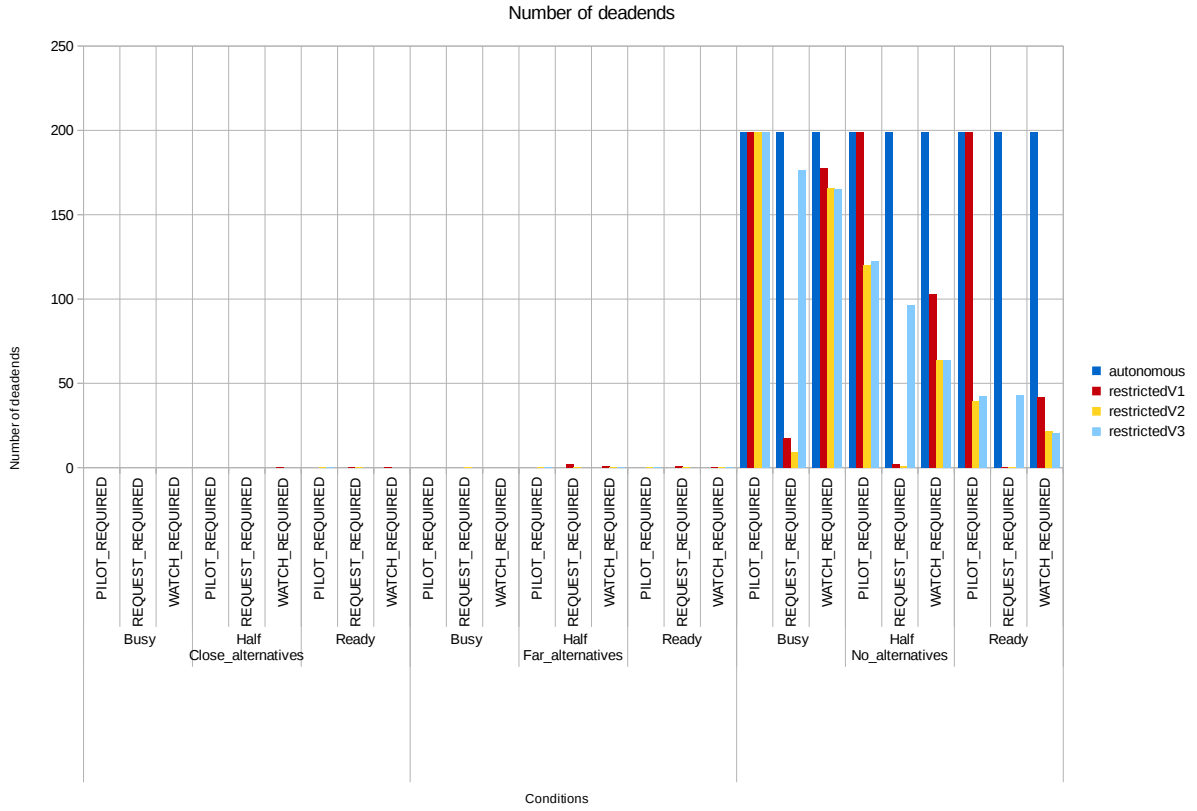


FIGURE 7.2 – Nombre de deadends atteints dans un environnement déterministe

.

Sans alternatives, les blocages sont évidemment plus présents. Les cas proches des 200 supposent que les 200 actions ont été atteintes avant d'accomplir l'objectif de la mission. L'agent autonome, n'étant pas capable d'interagir, était incapable de gérer la restriction. Il est donc resté en état de dead-end tout le long de ces expériences.

Nous pouvons observer qu'un agent appliquant la troisième méthode de restricted-MDP sera plus souvent bloqué en cas de restriction de type requête. Ceci est dû au fait que l'agent limite le nombre de requêtes envoyées à l'opérateur, tel qu'on peut le voir en figure 7.3. C'est la conséquence directe de demander à ce qu'un agent ne répète pas la même demande sans arrêt. Dans un cas général, où des alternatives sont présentes, l'agent parvient à s'en sortir, mais dans le cas où aucune restriction n'est disponible, cette contrainte est particulièrement pénalisante. Nous pouvons ainsi voir en figure 7.4 que l'agent suivant cette politique a plus de mal à accomplir sa mission qu'un agent suivant une autre des politiques. Au contraire, restrictedV2 peut répéter la même demande jusqu'à une moyenne de 10 fois par mission, lorsque l'opérateur est occupé. En dehors de ce type de restrictions, elle parvient à finir la mission plus rapidement que les autres dans le pire des cas.

Nous pouvons enfin observer que dans le cas où l'agent a plus d'alternatives, le nombre de demande s'élève surtout lorsque l'opérateur est disponible. En effet, dans le cas contraire, le coût d'intervention de l'opérateur est plus important que celui de contourner le problème.

Malgré tout, la figure 7.4 montre que l'impact des restrictions sur le temps d'exécution est léger avec des alternatives proches. Néanmoins, la présence d'un opérateur assez disponible a un

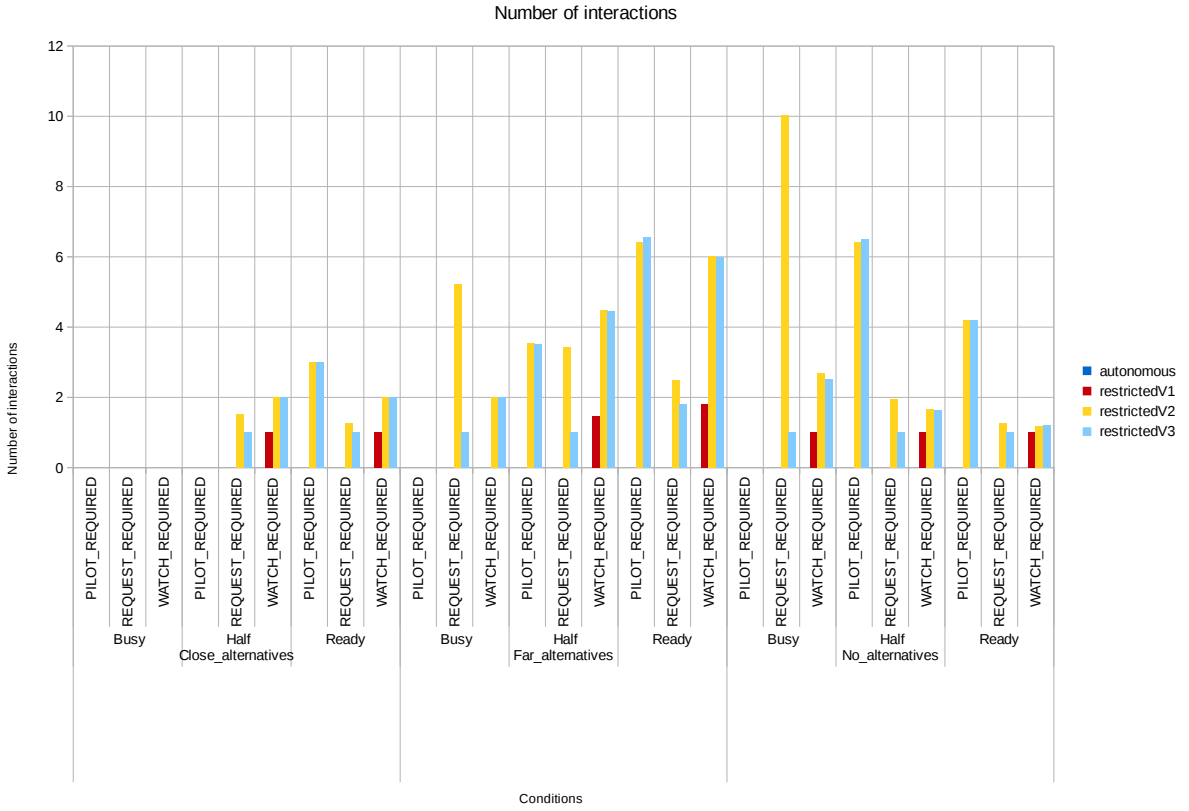


FIGURE 7.3 – Nombre d'interactions de la part de l'agent dans un environnement déterministe

.

fort impact sur les performances lorsque les alternatives sont très coûteuses voir inexistantes. L'impact est encore plus important pour une restriction de type requête, car l'opérateur pourra toujours répondre favorablement après un refus, excepté pour restrictedV3, puisque cette méthode consiste à ne pas répéter deux fois la même demande.

7.2.2 Environnement stochastique

Nous avons modélisé l'environnement stochastique de telle sorte que l'agent ait 10% de chances de prendre une direction autre que la direction voulue, tirée aléatoirement.

Les figures 7.5, 7.7, 7.6 présentent les résultats expérimentaux obtenus avec un environnement stochastique.

Nous pouvons constater que la durée de mission est presque réduite de moitié, à l'aide d'un opérateur disponible pour une restriction avec peu d'alternatives, néanmoins, l'impact est plus faible que dans le cadre déterministe. Néanmoins, de manière générale, la durée des missions avec des restrictions sans alternative semblent plus courtes dans le cadre stochastique que déterministe. Ceci est du au fait que la variation aléatoire de la position du robot lui permet de traverser la zone de restriction de manière "involontaire", permettant par exemple au robot d'aller au nord quand il prend la direction du sud.

Ces transitions involontaires permettent ainsi à l'agent d'éviter l'interaction avec l'opérateur. Par exemple, lorsqu'il n'y a pas d'alternatives, et que l'opérateur est occupé (ne répond favo-

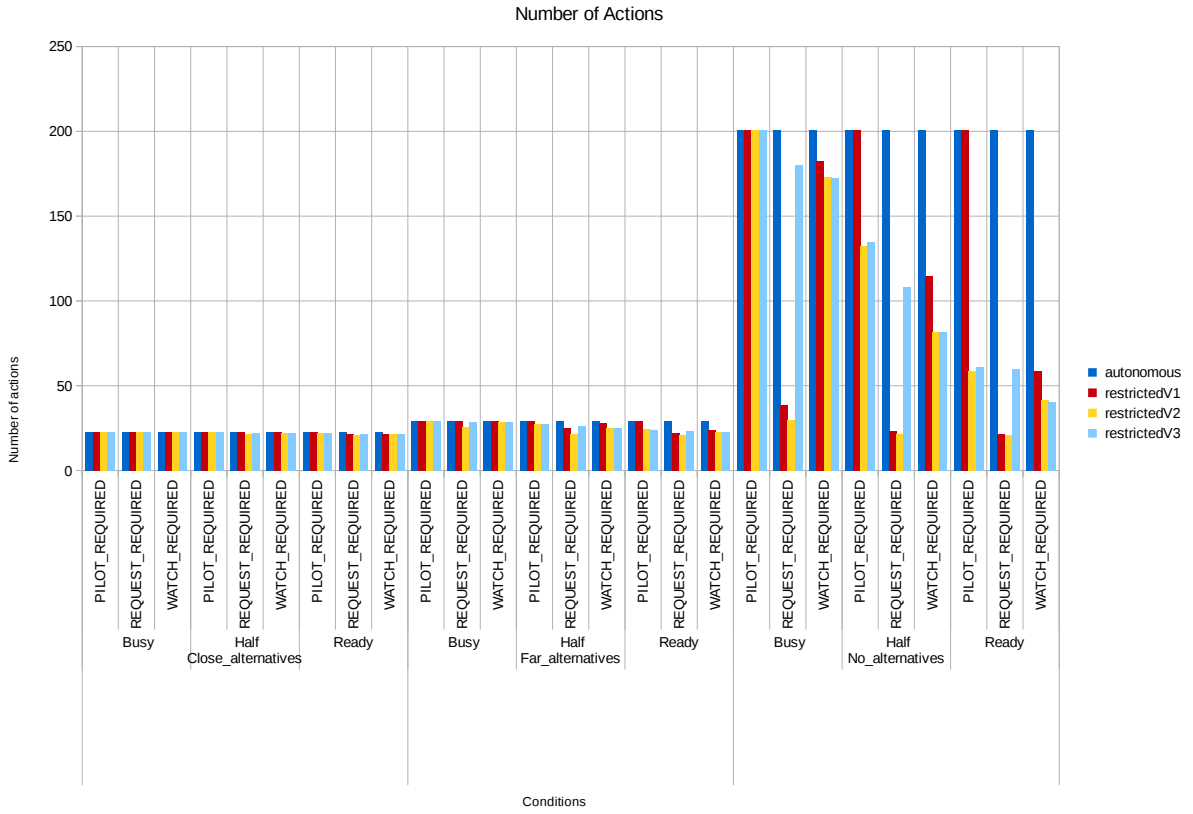


FIGURE 7.4 – Durée de la mission, exprimée en nombre d'actions, dans un environnement déterministe.

rablement qu'à 10% des requêtes, lorsqu'il est prêt), les méthodes restrictedV2 et restrictedV3 ne cherchent même pas à émettre une requête. C'est ainsi que ces méthodes, accompagnées de la méthode autonome, parviennent à accomplir l'objectif sans l'aide de l'opérateur. Néanmoins, elles prennent ainsi beaucoup plus de temps.

Ces déplacements involontaires entraînent aussi le robot à se déplacer en arrière, de temps en temps. L'agent repasse alors à un état déjà atteint auparavant, et entraînent donc la détection de deadends, tels qu'on en voit dans les cas où des alternatives sont disponibles. Cela peut aussi entraîner l'agent à se déplacer dans des zones restreintes, le menant à reculer puisqu'il ne répond pas aux conditions pour se déplacer dans cette zone.

On peut également noter que le nombre d'interactions est plus important dans un environnement stochastique que dans un environnement déterministe. Ceci s'explique d'une part car l'agent peut avoir plus de mal à "esquiver" une zone restreinte, et d'autre part car il peut être amené à retourner dans la zone restreinte après l'avoir dépassée.

Enfin, remarquons que, sans alternatives, la méthode restrictedV1 a tendance à émettre plus de requêtes que les autres, notamment pour un changement d'autonomie. Ceci est dû au fait qu'il ne demande le changement d'autorité que pour le moment où il traverse, sans anticipation, et sans en avoir le choix, lorsqu'il se retrouve par inadvertance dans une zone restreinte. Dans les autres cas, un agent animé par cette politique a plutôt tendance à chercher des alternatives, excepté pour une restriction de type `WATCH_REQUIRED`, moins coûteuse, et plus stable une fois obtenu.

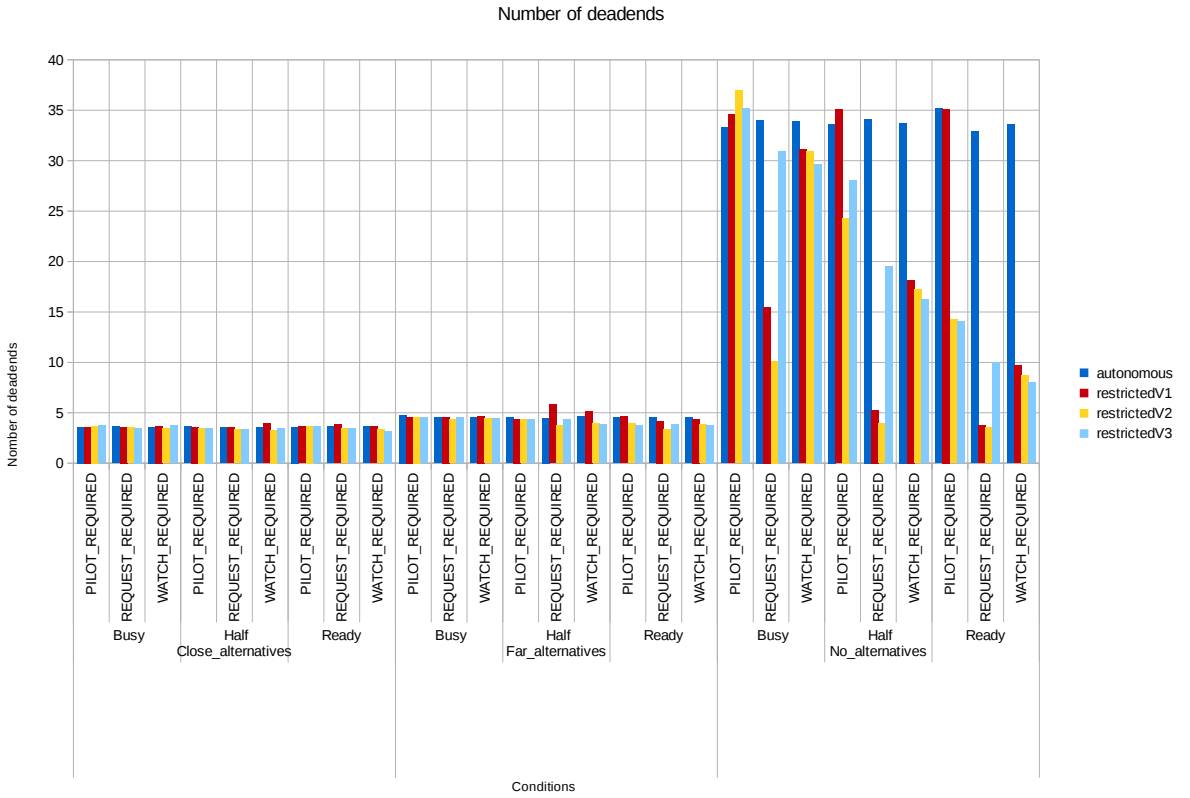


FIGURE 7.5 – Nombre de deadends atteints dans un environnement stochastique

.

7.3 Conclusion

Nous avons, au cours de ce chapitre, présenté la mise en place de nos résultats expérimentaux.

Dans un premier temps, nous avons parlé des conditions d'expériences dans lesquelles ces résultats ont été obtenus. Nous avons ainsi étudié l'impact du niveau de restriction, de la fréquence de réponse d'un opérateur, ainsi que des différentes alternatives à la restriction. Nous avons également considéré un environnement déterministe, puis stochastique.

Bien que les résultats soit préliminaires, ceux-ci ont pu démontrer l'intérêt de ces méthodes, permettant d'accélérer la mission lorsque un opérateur disponible a émis pour l'agent des restrictions difficiles à contourner. La 3ème méthode a montré de meilleurs résultats en général, mais son principe visant à ne pas répéter plusieurs fois la même demande bride ses performances, dans le cas d'une restriction de type `REQUEST_REQUIRED`, et devient moins rapide et moins fiable que la seconde. La première méthode permet quant à elle de limiter le nombre d'interactions entre l'opérateur et l'agent, mais au détriment des chances de succès de la mission.

Nous avons néanmoins pu constater que, dans un environnement stochastique, certains effets indésirables pouvaient avoir lieu. Par exemple, un agent ayant passé une restriction peut y retourner par malchance, ou peut traverser une zone restreinte dont il n'avait pas le droit, par chance. Néanmoins, ceci se traduit par une montée du nombre d'interactions, il semble donc que l'agent cherche malgré tout l'aide de l'opérateur quand ce dernier l'a jugé nécessaire.

Nous n'avons modélisé qu'une restriction à la fois, mais il serait intéressant de voir les

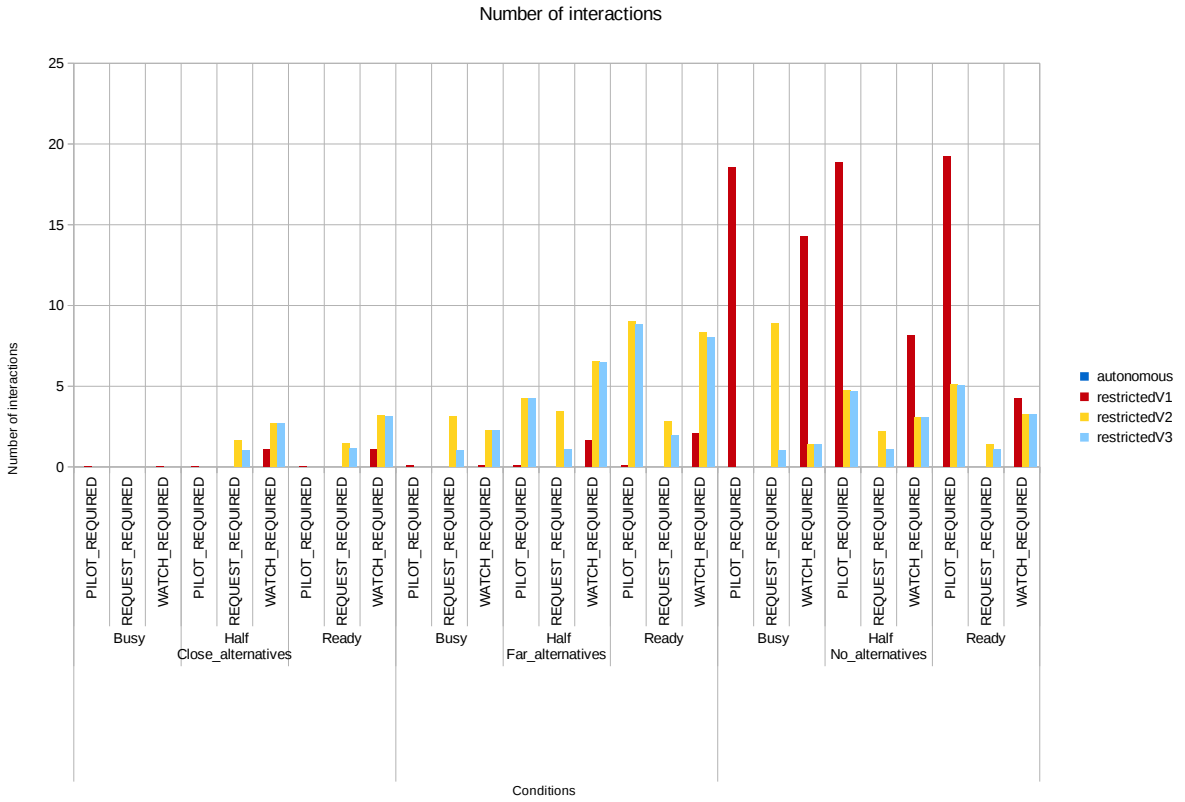


FIGURE 7.6 – Nombre d'interactions de la part de l'agent dans un environnement stochastique

.

réactions du robot face à plusieurs zones de restrictions. Nous pourrions nous demander, par exemple, si face à deux restrictions de niveau différents, l'agent serait en mesure de demander le niveau de restriction le plus élevé des deux, afin de ne pas se retrouver bloqué entre deux.

D'autre part, avec un modèle d'environnement stochastique différent, empêchant le robot de revenir en arrière par exemple, un agent pourrait peut-être voir son nombre d'interactions avec l'opérateur diminuer. Il faudrait également être en mesure de s'assurer que l'agent ne traverse pas par inadvertance des zones restreintes, sans autorisation.

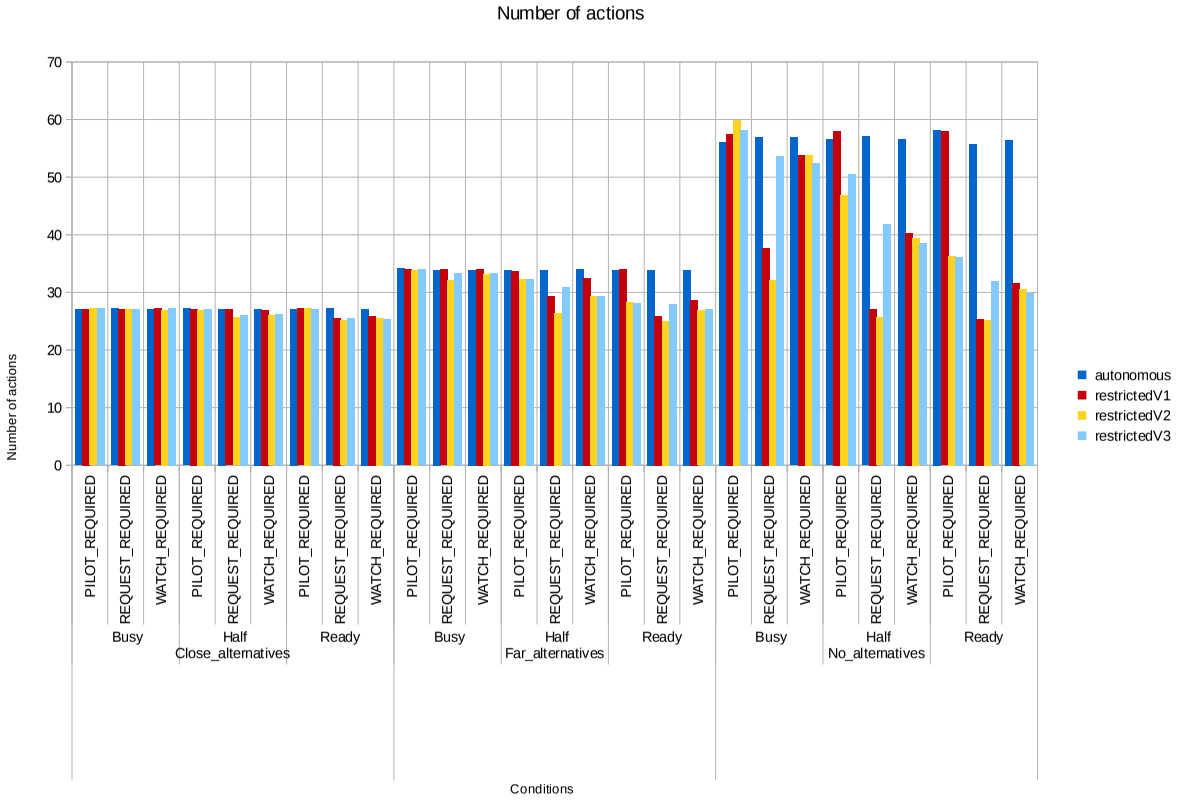


FIGURE 7.7 – Durée de la mission, exprimée en nombre d'actions, dans un environnement stochastique.

Conclusion de la Partie 3

Au cours de cette partie nous avons présenté les expérimentations qui nous ont permis d'évaluer les capacités des modèles développés au cours de cette thèse. Ces expériences concernaient deux problématiques : l'estimation de la politique d'un agent télé-opéré et la capacité d'un agent à respecter des restrictions d'un opérateur, et à interagir avec lui.

Estimation d'une politique d'un agent télé-opéré

Le chapitre 6 présente les expérimentations sur l'estimation d'une politique pour tester l'efficacité et la stabilité des algorithmes `FORCE`, `LEARN` et `DIST`, présentés en chapitre 4.

Nous avons dans un premier temps décrit les conditions d'expériences. Nous avons ensuite présenté un premier jeu de résultat, qui nous ont permis de voir que les solutions proposés amélioraient l'estimation de la politique de l'opérateur, notamment des méthodes `LEARN` et `FORCE` sur une mission répétée plusieurs fois, telle qu'une patrouille. Ces expériences ont tout de même permis de mettre en évidence une certaine instabilité des méthodes `LEARN` et `DIST`.

Nous avons donc proposé des améliorations pour diminuer ces instabilités, telles qu'une correction de `DIST`, une suppression d'objectifs parasites, ou la possibilité pour l'opérateur de renseigner des informations sur les états, telles que la proximité d'un mur à une position. Les expériences mises en place après ces mises à jours ont permis d'améliorer les performances et la stabilité de `LEARN` et de `DIST`, surtout sur une mission occasionnelle telle qu'un assaut. Malgré tout, ces méthodes ne permettent pas de surpasser une méthode de statistique, sur des missions régulièrement répétées, surtout dans des environnements stochastiques, et après un certain nombre de répétitions de la mission.

Ces résultats nous permettent de nous poser des questions. D'une part, le modèle de l'agent télé-opéré utilisé pour les expériences n'est pas idéal, car il ne prend pas en compte la capacité d'apprentissage de l'opérateur, diminuant ses hésitations et changeant ses préférences. En faisant des expériences sur un modèle plus fiable d'un humain, ou avec un vrai opérateur, les résultats pourraient varier. D'autre part, comme l'opérateur peut fournir des informations à l'agent sur les états, il serait intéressant de voir si les préférences apprises sur ces informations sont généralisables, permettant ainsi de dresser un profil d'utilisateur.

Nous avons pu voir au cours de nos expériences qu'il est parfois difficile de déterminer l'origine de la déviation de l'opérateur par rapport à la politique optimale, et donc quelle préférence est concernée. Il pourrait être intéressant, afin d'étudier cette lacune, d'étudier des concepts tels que l'affordance, le focus et le nimbus, présentés par [Zieba *et al.*, 2010], utilisés pour évaluer la capacité d'un agent à détecter ce qu'il va l'intéresser. D'autre part, nous avons modélisé les préférences de l'opérateur, mais pas les hésitations dans les politiques estimées. il pourrait être intéressant d'apprendre quand l'opérateur est susceptible d'hésiter, afin d'éviter de généraliser une hésitation ou de prévoir quand les actions de l'agent télé-opéré seront moins prévisibles.

Enfin, nous avons cherché à estimer la politique d'un agent télé-opéré pour qu'un agent autonome puisse se coordonner avec lui. Il reste à évaluer l'efficacité de ces estimations pour la coordination entre agents autonomes et télé-opérés.

Autonomie ajustable

Dans un second temps, nous avons étudié dans le chapitre 7 le comportement d'un agent appliquant une politique calculée par les modèles présentés en chapitre 5 visant à permettre à un opérateur de fixer des restrictions sur l'autonomie d'un agent au cours de la mission.

Nous avons commencé par définir les conditions d'expériences, et cherché à représenter les différentes situations dans lesquelles l'agent sera plus ou moins bloqué en fonction des restrictions qui lui seront imposées.

Nos expériences nous ont permis de démontrer que ces méthodes permettent à un agent de gagner en efficacité en interagissant avec l'opérateur lorsqu'il est disponible, par rapport à un agent devant suivre les restrictions mais devant rester autonome. La dernière méthode proposée permet d'obtenir un comportement proche de celui souhaité.

Néanmoins, il arrive tout de même à l'agent de se retrouver bloqué, en attendant que l'opérateur se montre plus disponible. Mais comme la connaissance de l'agent sur l'opérateur est fixe, celui-ci n'est pas en mesure de remettre en cause ses estimations sur son comportement, et donc revenir demander à l'opérateur de l'aide s'il a estimé auparavant qu'il valait mieux rester bloqué. Peut-être qu'une solution visant à demander automatiquement la télé-opération lors d'un blocage pourrait s'avérer intéressant.

Ces résultats n'ont été obtenus en n'appliquant qu'une restriction à la fois, ne permettant pas d'observer le comportement de l'agent pour faire face à plusieurs zones de restrictions. De plus, un environnement stochastique permet à un agent de passer outre ces restrictions via un déplacement aléatoire. Une fonction de transition plus réaliste permettrait peut-être de diminuer cet impact, néanmoins il faudrait s'assurer de diminuer ces risques au maximum.

Enfin, dans les deux contributions de cette thèse, nous avons proposés des solutions dans un cadre totalement observable. Néanmoins, les problèmes traités au cours de cette thèse sont encore plus présents dans un domaine partiellement observable.

Conclusion

Le travail dans cette thèse a pour objectif, d'une part, de permettre à des agents autonomes de se coordonner avec des agents télé-opérés, et d'autre part de permettre à un agent à autonomie ajustable de demander une variation de l'attention de son opérateur, en fonction des restrictions que ce dernier lui a imposé.

Nous avons travaillé dans un premier temps sur la coordination d'agents dans un système hétérogène. C'est à dire que tandis que la coordination d'agents autonomes peut se faire en s'adaptant à la politique des autres agents, il est difficile de le faire avec un agent piloté, car sa politique est inconnu. Nous cherchons donc à l'estimer, en prenant en compte que l'agent n'a pas toutes les connaissances sur les préférences de l'opérateur, ses connaissances, capacités de perception ou son modèle d'évaluation d'une situation (par exemple évaluation approximative d'une distance). Nous avons donc proposé des approches visant à apprendre la politique exécutée par l'opérateur lorsque celui-ci télé-opère le robot, robustes à ses hésitations. Là où les approches semblables nécessitent un apprentissage après la mission, nous proposons trois approche d'estimation en ligne de la politique. La première approche visait à retenir les dernières actions du robot à chaque état, et de calculer la politique optimale de ce robot en prenant en compte ces actions. La seconde méthode vise à apprendre les préférences de l'opérateur sur les états pour en déduire une politique en fonction de ces dernières. La dernière approche est tirée de la seconde, mais limite la propagation des préférences de l'opérateur pour éviter un sur-apprentissage. Nous avons ensuite proposé des améliorations de ces méthodes pour améliorer la stabilité et l'efficacité de l'estimation générée par ces méthodes.

Nos expériences ont démontré que nos algorithmes étaient efficaces et permettaient une nette amélioration de l'estimation de la politique d'un robot piloté. Bien que des méthodes, comme une approche statistique, peuvent parfois se montrer plus efficaces, c'est seulement après avoir répété la mission plusieurs fois, et surtout en environnement déterministe. De plus, en général, ces méthodes présentent une mauvaise estimation à la découverte de la misison. L'implémentation sur des robots a mis en évidence des limites dans nos méthodes de prédiction, comme l'apprentissage de déviations involontaires, dues à la précision de la télé-opération, ou l'incapacité d'apprendre les raisons d'une déviation si elle concerne un état éloigné du point de déviation.

On pourrait, suite à ces travaux, réfléchir à la manière de modéliser les hésitations de l'opérateur. En effet, un opérateur non-expert peut hésiter en cas d'imprévus en cours de mission, mais ce nombre d'hésitation devrait diminuer en fonction que l'opérateur s'adapte aux imprévus. D'autre part, il faudrait étudier la manière d'étudier la capacité d'un agent à se coordonner à partir des estimations générées. Nous pouvons également nous demander, puisque nous visons une coordination, s'il ne serait pas plus intéressant d'estimer une politique sous la forme de probabilité d'effectuer une action à un état donné, permettant ainsi de prendre en compte les cas où deux actions se valent en terme d'utilité. Enfin, nous devons compléter ce travail par la mise en place des algorithmes de résolution de jeux stochastiques (type "Best-Response") pour calculer les politiques coordonnées des agents autonomes avec celles estimées des agents télé-opérées.

Dans un second temps, nous avons étudié la possibilité d'intégrer de restrictions dans la politique d'un agent, visant à définir des niveaux minimums d'attention d'un opérateur en fonction de l'impact possible d'une action. Là où en temps normal l'autonomie ajustable sert à demander à un opérateur de prendre le relais lorsque l'agent est en difficulté, nous cherchons à permettre à l'opérateur de restreindre l'agent pour lui assurer un contrôle du robot, fiabiliser son comportement et définir soit-même les zones où le robot pourrait être en difficulté, par manque d'informations sur la zone par exemple.

Nous avons d'abord défini des niveaux de restrictions, associés à des niveau d'attention de la part de l'opérateur, comme la surveillance ou la télé-opération. Ensuite, nous avons proposé plusieurs approches. La première consiste à intégrer directement toutes les interactions avec l'opérateur dans la fonction de transition. Cela a pour avantage de simplifier la planification, mais l'agent n'est alors plus capable de faire une requête à l'avance, empêchant donc l'opérateur d'analyser la situation avant de prendre la main. Nous avons donc proposé une seconde méthode, intégrant les interactions sous la forme d'actions, choisies en parallèle des actions de contrôle du robot. Néanmoins, si l'opérateur refuse d'accorder le droit à un agent d'effectuer une action, alors en suivant ces deux premières méthodes, l'agent répétera la demande jusqu'à ce que l'opérateur accepte ou soit indisponible. Pour éviter ce blocage, nous proposons une troisième méthode, calculant une politique privée des demandes d'autorisations déjà refusées.

Nos premières expériences ont démontré que ces méthodes, associées à un opérateur accessible, permettent à un agent de gagner du temps en mission lorsque des restrictions sur son autonomie a été restreinte. La dernière méthode proposée est celle qui réduit le mieux ce temps d'exécution, excepté pour les cas où des restrictions de type `REQUEST_REQUIRED` lui ont été appliquées, étant donné qu'il a été bridé dans ces cas là. Mais bien qu'il soit moins efficace, il correspond d'avantage au comportement souhaité de l'opérateur.

Néanmoins, il arrive tout de même à l'agent de se retrouver bloqué, en attendant que l'opérateur se montre plus disponible. Mais comme la connaissance de l'agent sur l'opérateur est fixe, celui-ci n'est pas en mesure de remettre en cause ses estimations sur son comportement, et donc revenir demander à l'opérateur de l'aide s'il a estimé auparavant qu'il valait mieux rester bloqué. Peut-être qu'une solution visant à demander automatiquement la télé-opération lors d'un blocage pourrait s'avérer intéressant.

Ces résultats n'ont été obtenus en n'appliquant qu'une restriction à la fois, ne permettant pas d'observer le comportement de l'agent pour faire face à plusieurs zones de restrictions. De plus, un environnement stochastique permet à un agent de passer outre ces restrictions via un déplacement aléatoire. Une fonction de transition plus réaliste permettrait peut-être de diminuer cet impact, néanmoins il faudrait s'assurer de diminuer ces risques au maximum.

Enfin, dans les deux contributions de cette thèse, nous avons proposés des solutions dans un cadre totalement observable. Néanmoins, les problèmes traités au cours de cette thèse sont encore plus présents dans un domaine partiellement observable.

Bibliographie

[Abel *et al.*, 2017] Abel, D., Salvatier, J., Stuhlmüller, A. et Evans, O. (2017). Agent-Agnostic Human-in-the-Loop Reinforcement Learning. *arXiv :1701.04079 [cs]*. arXiv : 1701.04079.

[Alani *et al.*, 2003] Alani, H., Kim, S., Millard, D. E., Weal, M. J., Hall, W., Lewis, P. H. et Shadbolt, N. R. (2003). Automatic ontology-based knowledge extraction from web documents. *IEEE Intelligent Systems*, 18(1):14–21.

[Armstrong-Crews et Veloso, 2007] Armstrong-Crews, N. et Veloso, M. (2007). Oracular partially observable markov decision processes : A very special case. *In Robotics and Automation, 2007 IEEE International Conference on*, pages 2477–2482. IEEE.

[Bailey *et al.*, 2001] Bailey, B. P., Konstan, J. A. et Carlis, J. V. (2001). The effects of interruptions on task performance, annoyance, and anxiety in the user interface. *In Interact*, volume 1, pages 593–601.

[Baker et Yanco, 2004] Baker, M. et Yanco, H. A. (2004). Autonomy mode suggestions for improving human-robot interaction. *In Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 3, pages 2948–2953. IEEE.

[Barto *et al.*, 1983] Barto, A. G., Sutton, R. S. et Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846.

[Bellifemine *et al.*, 1999] Bellifemine, F., Poggi, A. et Rimassa, G. (1999). Jade–a fipa-compliant agent framework. *In Proceedings of PAAM*, volume 99, page 33. London.

[Bellman, 1956] Bellman, R. (1956). Dynamic programming and the smoothing problem. *Management Science*, 3(1):111–113.

[Bernstein *et al.*, 2000] Bernstein, D. S., Zilberstein, S. et Immerman, N. (2000). The complexity of decentralized control of markov decision processes. *In Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 32–37. Morgan Kaufmann Publishers Inc.

[Bertsekas *et al.*, 1995] Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P. et Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA.

[Bertsekas et Tsitsiklis, 1995] Bertsekas, D. P. et Tsitsiklis, J. N. (1995). Neuro-dynamic programming : an overview. *In Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE.

[Boutilier *et al.*, 1999] Boutilier, C., Dean, T. et Hanks, S. (1999). Decision-theoretic planning : Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11(1):94.

[Bradshaw *et al.*, 2003] Bradshaw, J. M., Sierhuis, M., Acquisti, A., Feltovich, P., Hoffman, R., Jeffers, R., Prescott, D., Suri, N., Uszok, A. et Van Hoof, R. (2003). Adjustable autonomy and human-agent teamwork in practice : An interim report on space applications. *Agent autonomy*, 20.

[Canu, 2011] Canu, A. (2011). *Planification multiagent sous incertitude orientée interactions : modèle et algorithmes*. Thèse de doctorat, Université de Caen.

[Carlson et Demiris, 2008] Carlson, T. et Demiris, Y. (2008). Human-wheelchair collaboration through prediction of intention and adaptive assistance. *In Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3926–3931. IEEE.

[Choi et Kim, 2011] Choi, J. et Kim, K.-E. (2011). Inverse reinforcement learning in partially observable environments. *Journal of Machine Learning Research*, 12(Mar):691–730.

[Clough, 2002] Clough, B. T. (2002). Metrics, schmetrics ! how the heck do you determine a uav's autonomy anyway. Rapport technique, AIR FORCE RESEARCH LAB WRIGHT-PATTERSON AFB OH.

[Cohen *et al.*, 2011] Cohen, R., Jung, H., Fleming, M. W. et Cheng, M. Y. (2011). A user modeling approach for reasoning about interaction sensitive to bother and its application to hospital decision scenarios. *User Modeling and User-Adapted Interaction*, 21(4-5):441–484.

[Cohen-Solal *et al.*, 2015] Cohen-Solal, Q., Bouzid, M. et Niveau, A. (2015). An algebra of granular temporal relations for qualitative reasoning. *In IJCAI*, pages 2869–2875.

[Cohn *et al.*, 2011] Cohn, R., Durfee, E. et Singh, S. (2011). Comparing action-query strategies in semi-autonomous agents. *In The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1287–1288. International Foundation for Autonomous Agents and Multiagent Systems.

[Collins *et al.*, 2000] Collins, J., Bilot, C. et Gini, M. (2000). Mixed-initiative decision support in agent-based automated contracting. *In Proceedings of the fourth international conference on Autonomous agents*, pages 247–254. ACM.

[Côté, 2013] Côté, N. (2013). *Mixed Decision and Reasoning for Adjustable Autonomy*. Theses, Université de Caen.

[Crandall *et al.*, 2005] Crandall, J. W., Goodrich, M. A., Olsen, D. R. et Nielsen, C. W. (2005). Validating human-robot interaction schemes in multitasking environments. *IEEE Transactions on Systems, Man, and Cybernetics-Part A : Systems and Humans*, 35(4):438–449.

[Dorais *et al.*, 1999] Dorais, G., Bonasso, R. P., Kortenkamp, D., Pell, B. et Schreckenghost, D. (1999). Adjustable autonomy for human-centered autonomous systems. *In Working notes of the Sixteenth International Joint Conference on Artificial Intelligence Workshop on Adjustable Autonomy Systems*, pages 16–35.

[Drougard *et al.*, 2015] Drougard, N., Teichteil-Konigsbuch, F., Farges, J.-L. et Dubois, D. (2015). Processus décisionnels de markov possibilités à observabilité mixte. *Revue des Sciences et Technologies de l'Information-Série RIA : Revue d'Intelligence Artificielle*, 29(6): pp–629.

[Eddy, 1996] Eddy, S. R. (1996). Hidden markov models. *Current opinion in structural biology*, 6(3):361–365.

[Enjalbert et Vanderhaegen, 2017] Enjalbert, S. et Vanderhaegen, F. (2017). A hybrid reinforced learning system to estimate resilience indicators. *Engineering Applications of Artificial Intelligence*, 64:295–301.

[Ferguson *et al.*, 1996] Ferguson, G., Allen, J. F., Miller, B. W. *et al.* (1996). Trains-95 : Towards a mixed-initiative planning assistant. *In AIPS*, pages 70–77.

[Fikes et Nilsson, 1971] Fikes, R. E. et Nilsson, N. J. (1971). Strips : A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208.

[Fleming et Cohen, 2004] Fleming, M. et Cohen, R. (2004). A decision procedure for autonomous agents to reason about interaction with humans. *In Proceedings of the AAAI 2004 Spring Symposium on Interaction between Humans and Autonomous Systems over Extended Operation*, pages 81–86.

[Fong *et al.*, 2001] Fong, T., Thorpe, C. et Baur, C. (2001). *Collaborative control : A robot-centric model for vehicle teleoperation*, volume 1. Carnegie Mellon University, The Robotics Institute.

[Fong *et al.*, 2003] Fong, T., Thorpe, C. et Baur, C. (2003). Robot, asker of questions. *Robotics and Autonomous systems*, 42(3-4):235–243.

[Guestrin *et al.*, 2002] Guestrin, C., Koller, D. et Parr, R. (2002). Multiagent planning with factored mdps. *In Advances in neural information processing systems*, pages 1523–1530.

[Hansen et Zilberstein, 2001] Hansen, E. A. et Zilberstein, S. (2001). Lao* : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62.

[He *et al.*, 2012] He, H., Eisner, J. et Daume, H. (2012). Imitation learning by coaching. *In* Pereira, F., Burges, C., Bottou, L. et Weinberger, K., éditeurs : *Advances in Neural Information Processing Systems 25*, pages 3149–3157. Curran Associates, Inc.

[Horvitz, 1999] Horvitz, E. (1999). Principles of mixed-initiative user interfaces. *In Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 159–166. ACM.

[Hüttenrauch et Severinson Eklundh, 2006] Hüttenrauch, H. et Severinson Eklundh, K. (2006). Beyond usability evaluation : Analysis of human-robot interaction at a major robotics competition. *Interaction Studies*, 7(3):455–477.

[International, 2016] International, S. (2016). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles.

[Ishikawa et Suzuki, 1997] Ishikawa, N. et Suzuki, K. (1997). Development of a human and robot collaborative system for inspecting patrol of nuclear power plants. *In Robot and Human Communication*, pages 118–123.

[Kennedy, 1999] Kennedy, C. M. (1999). Distributed reflective architectures for adjustable autonomy. *In International Joint Conference on Artificial Intelligence (IJCAI99), Workshop on Adjustable Autonomy.*

[Knox et Stone, 2008] Knox, W. et Stone, P. (2008). Tamer : Training an agent manually via evaluative reinforcement. *In Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*, pages 292–297.

[Knox et Stone, 2010] Knox, W. B. et Stone, P. (2010). Combining manual feedback with subsequent MDP reward signals for reinforcement learning. *In Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010).*

[Knox et Stone, 2012] Knox, W. B. et Stone, P. (2012). Reinforcement learning with human and mdp reward. *In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012).*

[Kraus, 1997] Kraus, S. (1997). Negotiation and cooperation in multi-agent environments. *Artificial intelligence*, 94(1-2):79–97.

[Le Guillarme, 2016] Le Guillarme, N. (2016). *A Game-Theoretic Planning Framework for Intentional Threat Assessment*. Thèse de doctorat, Université de Caen.

[Le Hy, 2007] Le Hy, R. (2007). *Programmation et apprentissage bayésien de comportements pour personnages synthétiques–application aux personnages de jeux vidéos*. Thèse de doctorat, Institut National Polytechnique de Grenoble-INPG.

[Lelerre et Mouaddib, 2016] Lelerre, M. et Mouaddib, A.-I. (2016). Non-optimal semi-autonomous agent behavior policy recognition. *In Proceedings of the 8th International Joint Conference on Computational Intelligence - Volume 1 : ECTA*, pages 193–200.

[Lelerre *et al.*, 2017] Lelerre, M., Mouaddib, A.-I. et JeanPierre, L. (2017). Robust inverse planning approaches for policy estimation of semi-autonomous agents. *In International Conference on Tools for Artificial intelligence*.

[Lesser *et al.*, 1999] Lesser, V., Atighetchi, M., Benyo, B., Horling, B., Raja, A., Vincent, R., Wagner, T., Xuan, P. et Zhang, S. X. (1999). The intelligent home testbed. *environment*, 2:15.

[Lesser, 1999] Lesser, V. R. (1999). Cooperative multiagent systems : A personal view of the state of the art. *IEEE Transactions on knowledge and data engineering*, 11(1):133–142.

[Littman, 1996] Littman, M. L. (1996). Algorithms for sequential decision making.

[Lopes *et al.*, 2009] Lopes, M., Melo, F. et Montesano, L. (2009). Active learning for reward estimation in inverse reinforcement learning. *In Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer.

[Madani *et al.*, 1999] Madani, O., Hanks, S. et Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable markov decision problems. *In AAAI/IAAI*, pages 541–548.

[Mercier, 2011] Mercier, S. (2011). *Authority sharing control among heterogeneous agents*. Theses, Ecole nationale superieure de l'aeronautique et de l'espace.

[Monderer et Shapley, 1996] Monderer, D. et Shapley, L. S. (1996). Potential games. *Games and economic behavior*, 14(1):124–143.

[Mooney et Bunescu, 2005] Mooney, R. J. et Bunescu, R. (2005). Mining knowledge from text using information extraction. *ACM SIGKDD explorations newsletter*, 7(1):3–10.

[Moore et Atkeson, 1993] Moore, A. W. et Atkeson, C. G. (1993). Prioritized sweeping : Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130.

[Mouaddib *et al.*, 2015] Mouaddib, A.-I., Jeanpierre, L. et Zilberstein, S. (2015). Handling advice in mdps for semi-autonomous systems. *In ICAPS Woskhop on Planning and Robotics (PlanRob)*, pages 153–160.

[Mouaddib *et al.*, 2010a] Mouaddib, A.-I., Zilberstein, S., Beynier, A. et Jeanpierre, L. (2010a). A decision-theoretic approach to cooperative control and adjustable autonomy. *In Proceedings of the 2010 Conference on ECAI 2010 : 19th European Conference on Artificial Intelligence*, pages 971–972, Amsterdam, The Netherlands, The Netherlands. IOS Press.

[Mouaddib *et al.*, 2010b] Mouaddib, A.-I., Zilberstein, S., Beynier, A. et Jeanpierre, L. (2010b). A decision-theoretic approach to cooperative control and adjustable autonomy. *In Proceedings of the 2010 conference on ECAI 2010 : 19th European Conference on Artificial Intelligence*, pages 971–972. IOS Press.

[Nair *et al.*, 2003] Nair, R., Tambe, M., Yokoo, M., Pynadath, D. V. et Marsella, S. (2003). Taming decentralized pomdps : Towards efficient policy computation for multiagent

settings. *In IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 705–711.

[Ng et Russell, 2000] Ng, A. Y. et Russell, S. (2000). Algorithms for inverse reinforcement learning. *In in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann.

[Ng *et al.*, 2000] Ng, A. Y., Russell, S. J. *et al.* (2000). Algorithms for inverse reinforcement learning. *In Icml*, pages 663–670.

[O'Brien et Nicol, 1998] O'Brien, P. D. et Nicol, R. C. (1998). Fipa—towards a standard for software agents. *BT Technology Journal*, 16(3):51–59.

[Olsen et Goodrich, 2003] Olsen, D. R. et Goodrich, M. A. (2003). Metrics for evaluating human-robot interactions. *In Proceedings of PERMIS*, volume 2003, page 4.

[Panagou et Kumar, 2014] Panagou, D. et Kumar, V. (2014). Cooperative Visibility Maintenance for Leader-Follower Formations in Obstacle Environments. *Robotics, IEEE Transactions on*, 30(4):831–844.

[Papadimitriou et Tsitsiklis, 1987] Papadimitriou, C. H. et Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450.

[Paruchuri *et al.*, 2008] Paruchuri, P., Pearce, J. P., Marecki, J., Tambe, M., Ordonez, F. et Kraus, S. (2008). Playing games for security : An efficient exact algorithm for solving bayesian stackelberg games. *In Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '08, pages 895–902, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Pashenkova *et al.*, 1996] Pashenkova, E., Rish, I. et Dechter, R. (1996). Value iteration and policy iteration algorithms for markov decision problem. *In AAAI'96 : Workshop on Structural Issues in Planning and Temporal Reasoning*. Citeseer.

[Peng et Williams, 1993] Peng, J. et Williams, R. J. (1993). Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 1(4):437–454.

[Picard *et al.*, 2017] Picard, G., Balbo, F. et Boissier, O. (2017). Approches multiagents pour l'allocation de courses à une flotte de taxis autonomes. *In 25es Journées Francophones sur les Systèmes Multi-Agents (JFSMA)*, page to appear, Caen, France. Cépaduès.

[Puterman, 1994] Puterman, M. L. (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st édition.

[Rao et Georgeff, 1991] Rao, A. S. et Georgeff, M. P. (1991). Modeling rational agents within a bdi-architecture. *KR*, 91:473–484.

[Riley et Veloso, 2004] Riley, P. et Veloso, M. M. (2004). Advice generation from observed execution : Abstract markov decision process learning. *In AAAI*, pages 631–637.

[Rosenthal et Veloso, 2011] Rosenthal, S. et Veloso, M. (2011). Modeling humans as observation providers using pomdps. *In RO-MAN, 2011 IEEE*, pages 53–58. IEEE.

[Rosenthal *et al.*, 2012] Rosenthal, S., Veloso, M. et Dey, A. K. (2012). Is someone in this office available to help me ? *Journal of Intelligent & Robotic Systems*, 66(1-2):205–221.

[Russell et Norvig, 2003] Russell, S. J. et Norvig, P. (2003). *Artificial Intelligence : A Modern Approach*. Pearson Education, 2 édition.

[Scerri *et al.*, 2002] Scerri, P., Pynadath, D. V. et Tambe, M. (2002). Towards adjustable autonomy for the real world. *Journal of Artificial Intelligence Research*, 17(1):171–228.

[Schaal *et al.*, 2003] Schaal, S., Ijspeert, A. et Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society B : Biological Sciences*, 358(1431):537–547.

[Schmidhuber, 2015] Schmidhuber, J. (2015). Deep learning in neural networks : An overview. *Neural networks*, 61:85–117.

[Shiomi *et al.*, 2008] Shiomi, M., Sakamoto, D., Kanda, T., Ishi, C. T., Ishiguro, H. et Hagita, N. (2008). A semi-autonomous communication robot : a field trial at a train station. *In Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction*, pages 303–310, New York, NY, USA. ACM.

[Sigaud et Buffet, 2010] Sigaud, O. et Buffet, O. (2010). *Markov Decision Processes in Artificial Intelligence*. Wiley-ISTE.

[Sigaud et Buffet, 2013] Sigaud, O. et Buffet, O. (2013). *Markov decision processes in artificial intelligence*. John Wiley & Sons.

[Sprauel *et al.*, 2014] Sprauel, J., Kolobov, A. et Teichteil-Königsbuch, F. (2014). Saturated Path-Constrained MDP : Planning under Uncertainty and Deterministic Model-Checking Constraints. *In AAAI*, pages 2367–2373.

[Sutton, 1991] Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163.

[Sutton et Barto, 1998a] Sutton, R. S. et Barto, A. G. (1998a). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA.

[Sutton et Barto, 1998b] Sutton, R. S. et Barto, A. G. (1998b). *Reinforcement learning : An introduction*, volume 1. MIT press Cambridge.

[Tambe, 1997] Tambe, M. (1997). Towards flexible teamwork. *Journal of artificial intelligence research*, 7:83–124.

[Tambe, 2008] Tambe, M. (2008). Electric elves : What went wrong and why. *AI magazine*, 29(2):23.

[Taylor, 2003] Taylor, R. M. (2003). Capability, cognition and autonomy. Rapport technique, QINETIQ LTD FARNBOROUGH (UNITED KINGDOM) CENTRE FOR HUMAN SCIENCES.

[Thrun, 1992] Thrun, S. B. (1992). The role of exploration in learning control.

[Vorobeychik *et al.*, 2012] Vorobeychik, Y., An, B. et Tambe, M. (2012). Adversarial patrolling games. *In Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '12, pages 1307–1308, Richland, SC. International Foundation for Autonomous Agents and Multiagent Systems.

[Vorobeychik et Singh, 2012] Vorobeychik, Y. et Singh, S. P. (2012). Computing stackelberg equilibria in discounted stochastic games. *In AAAI*.

[Watkins et Dayan, 1992a] Watkins, C. et Dayan, P. (1992a). Q-learning. *Machine Learning*, 8(3-4):279–292.

[Watkins et Dayan, 1992b] Watkins, C. J. et Dayan, P. (1992b). Q-learning. *Machine learning*, 8(3-4):279–292.

[Weiss, 1960] Weiss, G. (1960). Dynamic programming and markov processes. ronald a. howard. technology press and wiley, new york, 1960. viii + 136 pp. illus. $5.75. *Science*, 132(3428):667–667.

[Weiss, 1999] Weiss, G. (1999). *Multiagent systems : a modern approach to distributed artificial intelligence*. MIT press.

[Zieba *et al.*, 2010] Zieba, S., Polet, P., Vanderhaegen, F. et Debernard, S. (2010). Principles of adjustable autonomy : a framework for resilient human–machine cooperation. *Cognition, Technology & Work*, 12(3):193–203.

[Ziebart *et al.*, 2008] Ziebart, B. D., Maas, A. L., Bagnell, J. A. et Dey, A. K. (2008). Maximum Entropy Inverse Reinforcement Learning. *In AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA.

[Zilberstein, 2015] Zilberstein, S. (2015). Building Strong Semi-Autonomous Systems. *In AAAI*, pages 4088–4092.

Processus Décisionnels de Markov pour l'autonomie ajustable et l'interaction hétérogène entre engins autonomes et pilotés

Les robots vont être de plus en plus utilisés dans les domaines civils, comme dans le domaine militaire. Ces robots, opérant en flottes, peuvent accompagner des soldats au combat, ou accomplir une mission en étant supervisés par un poste de contrôle. Du fait des exigences d'une opération militaire, il est difficile de laisser les robots décider de leurs actions sans accord d'un opérateur ou surveillance, en fonction de la situation. Dans cette thèse, nous nous attardons sur deux problématiques :

D'une part, nous cherchons à exploiter l'autonomie ajustable de sorte à ce qu'un robot puisse accomplir sa mission de la manière la plus efficace possible, tout en respectant des restrictions assignées par un opérateur sur son niveau d'autonomie. Pour cela, celui-ci est en mesure de définir pour un ensemble d'états et d'actions donné un niveau de restriction. Ce niveau peut par exemple imposer au robot la télé-opération pour accéder à une zone à risque.

D'autre part, comme nous envisageons la possibilité que plusieurs robots soient déployés en même temps, ces robots doivent se coordonner pour accomplir leurs objectifs. Seulement, comme les opérateurs peuvent prendre le contrôle de certains d'entre eux, la question de la coordination se pose. En effet, l'opérateur ayant ses propres préférences, perception de l'environnement, connaissances et étant sujet aux stress, hésitations, il est difficile de prévoir les actions que celui-ci va effectuer, et donc de s'y coordonner. Nous proposerons dans cette thèse une approche visant à estimer la politique exécutée par un robot télé-opéré à partir d'apprentissage basé sur les actions observés de ce robot.

La notion de planification est très présente dans ces travaux. Ceux-ci se baseront sur des modèles de planifications comme les Processus Décisionnels de Markov.

Mots-clés : planification sous incertitude, MDP, apprentissage, autonomie ajustable, systèmes semi-autonomes

Markov Decision Processes for adjustable autonomy and heterogeneous interaction between autonomous and piloted robots

Robots will be more and more used in both civil and military fields. These robots, operating in fleet, can accompany soldiers in fight, or accomplish a mission while being supervised by a control center. Considering the requirement of a military operation, it is complicated to let robots decide their action without an operator agreement or watch, in function of the situation.In this thesis, we focus on two problematics :

First, we try to exploit adjustable autonomy to make a robot accomplishes is mission as efficiency as possible, while he respects restrictions, assigned by an operator, on his autonomy level. For this, it is able to define for given sets of states and actions a restriction level. This restriction can force, for example, the need of being tele-operated to access a dangerous zone.

Secondly, we consider that several robots can be deployed at the same time. These robots have to coordinate to accomplish their objectives. However, since operators can take the control of some robots, the coordination is harder. In fact, the operator has preferences, perception, hesitation, stress that are not modeled by the agent. It is then hard to estimate his next actions, so to coordinate with him. We propose in this thesis an approach to estimate the policy executed by a tele-operated robot from learning methods, based on observed actions from this robot.

The notion of planning his important in these works. These are based on planning models, such as Markov Decision Processes.

Keywords : Planning under uncertainty, MDP, learning, adjustable autonomy, semi-autonomous system