MAIA : Machine Intelligente Autonome

Béchu Jérôme
eXia A5
2008

Rapport de stage

Apprentissage par Renforcement pour un Robot Mobile

Maître de stage : Alain Dutech

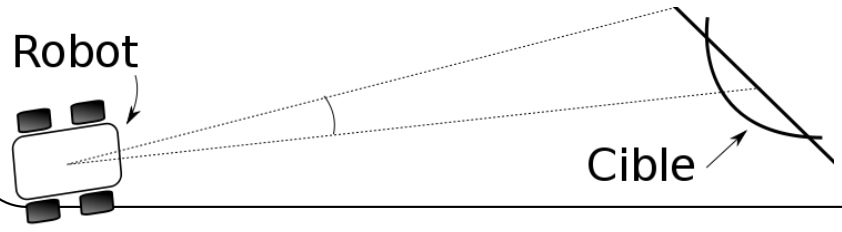
MAIA : MACHINE
INTELLIGENTE
AUTONOME

BÉCHU JÉRÔME
eXia A5
2008

RÉSUMÉ :

Le sujet du stage est l'implémentation d'un mécanisme d'apprentissage sur un **robot** pour lui permettre d'apprendre à se diriger vers une **cible**. Pour atteindre cet objectif on appliquera deux phases : *l'apprentissage tutoré* et *l'apprentissage libre*. Durant la phase d'apprentissage tutoré, un utilisateur indiquera les mouvements adéquats au robot pour que celui-ci atteigne la cible. Durant cet première phase d'apprentissage le robot doit être capable d'évaluer sa position par rapport à la cible afin d'être en mesure d'exploiter ces données. Durant la seconde phase, l'apprentissage libre, le robot doit être en mesure de localiser la cible à l'aide de sa caméra puis de choisir le mouvement approprié pour s'approcher de cette cible. Dans ce contexte deux sous problématiques se dégagent :

– Traiter des signaux continus
– Apprendre avec peu d'exemple



MOTS-CLEFS :

Q-learning, traces d'éligibilité, wifibot, Hedger, Peng, GSL, valeurs singulières, matrice pseudo inverse

Remerciements

AVANT toute rédaction, il me parait opportun de remercier toutes les personnes qui m'ont beaucoup appris et aidé lors de ce stage.

Je tiens à remercier avant tout Alain Dutech, mon maître de stage, qui m'a aiguillé tout au long du stage (et accessoirement m'a aidé à obtenir un emploi ◡̈). Cette expérience m'a beaucoup apporté du point de vue des compétences scientifiques (intelligence artificielle, linux) et à titre personnel.

J'aimerai également remercier toute l'équipe MAIA et notamment Olivier Buffet, Vincent Thomas ainsi que François Charpillet. Merci également à Olivier Rochel pour ses précieux conseils en développement.

Merci également à mes re-lectrices préférées : Marie-Edwige et Nadège.

Je tiens également à dire merci à mes compagnons de bureau : Nicolas, Antoine, Guillaume et Mahuna pour leurs encouragements et leur aide.

Enfin mes derniers remerciements sont destinés à mon ami, Xavier, pour son soutien inconditionnel !

Table des matières

Liste des Algorithmes

Chapitre 1

Introduction

L'intelligence artificielle est la discipline informatique qui doit permettre aux systèmes informatiques de réaliser des tâches qui sont à l'heure actuelle réalisées par l'homme. Ces tâches requièrent un raisonnement trivial pour l'homme, mais qui peut se révéler très complexe pour une intelligence artificielle. Dans ce contexte, l'équipe MAIA (MAchine Intelligente Autonome) qui m'a offert ce stage, s'intéresse à la modélisation, la construction et la simulation de systèmes composés d'un ou plusieurs agents.

Le sujet du stage est l'implémentation d'un mécanisme d'apprentissage sur un **robot** pour lui permettre d'apprendre à se diriger vers une **cible**. Pour atteindre cet objectif on appliquera deux phases : *l'apprentissage tutoré* et *l'apprentissage libre*. Durant la phase d'apprentissage tutoré, un utilisateur indiquera les mouvements adéquats au robot pour que celui-ci atteigne la cible. Durant cet première phase d'apprentissage le robot doit être capable d'évaluer sa position par rapport à la cible afin d'être en mesure d'exploiter ces données. Durant la seconde phase, l'apprentissage libre, le robot doit être en mesure de localiser la cible à l'aide de sa caméra puis de choisir le mouvement approprié pour s'approcher de cette cible. Il sera donc nécessaire de doter un robot de capacités d'apprentissage de ses comportements afin d'atteindre un objectif déterminé. Ce rapport sera tout d'abord composé d'une présentation du loria ; puis nous développerons les attentes du projet, le formalisme nécessaire et enfin nous discuterons des résultats obtenus.

Chap 2 : Présentation du Loria Ce chapitre présente le laboratoire de recheche lorain et plus particulièrement l'équipe MAIA.

Chap 3 : Spécifications fontionnelles Nous aborderons dans ce chapitre les fonctions essentielles de chaque phase du projet.

Chap 4 : Formalisme Pour comprendre le principe même du projet, il est primordial de disposer de quelques bases théoriques que nous offrirons dans ce chapitre.

Chap 5 : Spécifications techniques Dans ce chapitre nous expliquerons le fonctionnement des algorithmes que nous allons utiliser et des composantes logicielles nécessaire à ceux-ci.

Chap 6 : Le projet APRAM Ce chapitre va contenir la liste des tâches, phases et sous phases du projet. Il comprendra un planning de ces tâches.

Chap 7 : Les livrables Nous présenterons les «livrables» finaux et nous ferons un constat entre les attentes initiales et le résultat obtenu.

Chap 8 : Conclusion Pour clôre ce rapport, ce chapitre va présenter les conclusions et perspectives du projet.

Chapitre 2

Présentation du LORIA

2.1 INRIA

L'institut national de recherche en informatique et automatique est un organisme public civil de recherche français crée le 3 janvier 1967 suite au lancement du plan Calcul. Son ambition est de mettre en réseau les compétences et talents de l'ensemble du dispositif de recherche français, dans le domaine des sciences et technologies de l'information.

2.2 LORIA

2.2.1 Présentation succinte

Le LORIA, Laboratoire Lorrain de Recherche en Informatique et ses Applications, est une Unité Mixte de Recherche - UMR 7503 - commune à plusieurs établissements :

- CNRS, Centre National de Recherche Scientifique
- INPL, Institut National Polytechnique de Lorraine
- INRIA, Institut National de Recherche en Informatique et en Automatique
- UHP, Université Henri Poincaré, Nancy 1
- Nancy 2, Université Nancy 2

La création de cette unité a été officialisée le 19 décembre 1997 par la signature du contrat quadriennal avec le Ministère de l'Education Nationale, de la Recherche et de la Technologie et par une convention entre les cinq partenaires. Cette unité, renouvelée en 2001, succède ainsi au CRIN (Centre de Recherche en Informatique de Nancy), et associe les équipes communes entre celui-ci et l'Unité de Recherche INRIA Lorraine.

Le LORIA est un Laboratoire de plus de 450 personnes parmi lesquelles

- 150 chercheurs et enseignants-chercheurs
- un tiers de doctorants et post doctorants
- des ingénieurs, techniciens et personnels administratifs

organisé en équipes de recherche et services de soutien à la recherche

C'est aussi chaque année

- une trentaine de chercheurs étrangers invités
- des coopérations internationales avec des pays des cinq continents
- une quarantaine de contrats industriels

Missions

- Recherche fondamentale et appliquée au niveau international dans le domaine des Sciences et Technologies de l'Information et de la Communication
- Formation par la recherche en partenariat avec les Universités lorraines
- Transfert technologique par le biais de partenariats industriels et par l'aide à la création d'entreprises

2.2.2 Thématiques de recherche

Le LORIA souhaite développer ses recherches au niveau international dans les thématiques suivantes. Pour chacune, des équipes sont impliquées et des défis scien-

tifiques sont identifiés.

Calculs, simulation et visualisation à haute performance

La physique, la chimie ou la biologie soulèvent aujourd hui des problèmes de modélisation qui constituent de vrais défis pour l informatique : la taille des modèles numériques à calculer, transférer et visualiser, le besoin impérieux de précision géométrique et physique, la prise en compte de la composante temporelle et de l interactivité dans les techniques de simulation et de visualisation.

Qualité et sûreté des logiciels

Les problèmes de fiabilité et de sécurité des systèmes informatiques sont abordés en se focalisant sur deux objectifs : 1) la conception de logiciels sûrs et permettant le traitement d applications de taille réelle – 2) la vérification des systèmes et services critiques, embarqués ou enfouis.

Systèmes parallèles, distribués et communicants

Calculer sur des plates-formes hétérogènes distribuées, maîtriser le fonctionnement des réseaux hétérogènes et complexes, nécessite des recherches et des développements autour de modèles et d'architectures logiciels pour la surveillance des services à forte valeur ajoutée, d architectures d applications, entre autres pour la grille, les entreprises virtuelles ou l intelligence ambiante. Les compétences du LORIA portent sur la supervision, la coopération, et la distribution des programmes et des services, mais aussi sur l étude de protocoles, la sûreté de fonctionnement, et l évaluation de performances.

Modèles et algorithmes pour les sciences du vivant

En bioinformatique, un des objectifs de recherche est d expliquer comment les composants d un système biologique interagissent pour assurer une certaine fonction ; un modèle informatique permet d analyser, de simuler et à terme de prédire le comportement d un système biologique ainsi que de raisonner avec des méthodes formelles sur les propriétés de ce système. Un autre objectif est de mieux comprendre les mécanismes d expression de génomes à partir de leurs séquences nucléotidiques ; cela nécessite la mise au point de nouvelles méthodes algorithmiques efficaces, tenant compte d une prolifération de données génomiques et de nouvelles connaissances biologiques. L informatique bio-inspirée s appuie sur la conception de systèmes multi-agents, de systèmes neuromimétiques continus ou à évènements discrets, de systèmes connexionistes de très grande taille. L objectif est d étudier la capacité d apprentissage de ces systèmes bio-inspirés, grâce à des structures et algorithmes performants alliant méthodes mathématiques et connaissances biologiques.

Traitement de la langue naturelle et communication multimodale

Le LORIA rassemble des compétences en traitement automatique de la parole, en informatique linguistique, en interaction homme-machine, en vision, en analyse de

documents, en simulation de processus perceptifs et communicatifs. Cette convergence permet de s attaquer à l interprétation et à la présentation de données multimodales, avec pour objectif de modéliser la perception et la cognition humaines et d en tirer parti pour mieux communiquer et interagir avec l utilisateur.

Représentation et gestion des connaissances

Si les capacités de stockage et d organisation des systèmes d' information permettent d envisager des masses de données gigantesques, encore faut-il se doter des moyens de les exploiter et de les partager, au travers d outils d indexation, de navigation, de recherche d informations, de coopération. Cette notion soulève des problématiques nombreuses, liées au caractère très hétérogène des documents manipulés et aux critères d organisation des données qui peuvent être très variables suivant les organisations concernées. La représentation et la gestion des connaissances s attaquent à ces problèmes complexes, dans le cadre du Web sémantique et des entreprises virtuelles.

2.3 Les équipes de recherche

De nombreuses équipes de recherche exercent au Loria, en voici la liste exhaustive :

ABC Machine Learning and Computational Biology (LORIA Team)

ADAGIo Discrete Algorithmic and its Applications to Genomics and Imagery

ALGORILLE Algorithms for the Grid

ALICE Geometry and Lighting

CACAO

CALLIGRAMME Linear Logic, Demonstration Networks and Categoric Grammars

CASSIS Combination of Approaches to the Security of Infinite States Systems

CORTEX Neuromimetic intelligence

DEDALE Specification Development

ECOO Environments for COOperation

MACSI Modelling, Analysis and Control of Industrial Systems

MADYNES Management of Dynamic Networks and Services

MAGRIT Augmentation visuelle d'environnements complexes

MAIA MAchine Intelligente Autonome

MERLIN Méthodes pour l'Ergonomie des Logiciels Interactifs

MOSEL Formal Methods and Applications

ORPAILLEUR Knowledge representation, reasoning, knowledge discovery from databases

PAROLE Analysis, perception and automatic recognition of speech

PROTHEO Constraints, automated deduction and proofs of software properties

QGAR Querying Graphics through Analysis and Recognition

READ Recognition of writing and analysis of documents

SITE Modélisation et Développement de Systèmes d'Intelligence Économique

SPACES Solving Problems through Algebraic Computation and Efficient Software

TALARIS Traitement Automatique des Langues : Représentations, Inférences et Sémantique

TRIO Real Time and InterOperability

TYPES Logic, Proof Theory and Programming

VEGAS Effective Geometric Algorithms for Surfaces and Visibility

CARTE

2.4 MAIA

http ://www.loria.fr/equipes/maia/

Notre problématique s'inscrit dans le domaine de l'intelligence artificielle (I.A) dont les deux approches complémentaires nous permettent d'aborder notre objectif : soit imiter le comportement d' entités biologiques dans leurs aspects intelligents, soit élaborer des modèles qui permettent de doter un ou plusieurs agents de capacités habituellement attribuées à l intelligence sans forcément pour autant s inspirer des processus cognitifs sous-jacents. Dans ce contexte, nos travaux concernent la conception d entités informatiques (agents) capables de percevoir l'environnement, de l'interpréter et d'agir sur celui-ci avec une relative autonomie.

Axes thématiques :

- L intelligence artificielle distribuée : étude des phénomènes d'interaction et d'organisation, pilotage d'algorithmes et d'agents d'interprétation, simulation, résolution de problèmes
- La résolution de problèmes sous contraintes de ressources : conception, modélisation et pilotage d algorithmes "anytime"
- Les modèles de décision stochastique pour la planification et la perception : modèles de décisions markoviens (MDP), modèles de décisions markoviens partiellement observables (POMDP)
- L'interprétation de signaux industriels et médicaux

Relations scientifiques et industrielles

- Participation aux programmes Esprit (projets AITRAS, REAKT, REAKTANSE)
- Conventions avec EDF, IRSID, DCN Toulon
- Participation à l action incitative de la direction scientifique de l INRIA sur la résolution de problème avec limitation de ressources (avec les projets OMEGA, ORION et SHERPA)
- Participation à un projet NSF INRIA avec l équipe "Resource Bounded Reasoning Research Group" de l Université de Massachusetts at Amherst
- Participation au GdR-PRC Information Interaction et Intelligence
- Participation au GIS Sciences de la Cognition en collaboration avec le Laboratoire de Biologie et Physiologie du comportement de l UHP et le Laboratoire d Éthologie de Rennes 1
- Collaboration avec la NASA, CHU, ALTIR
- Participation au projet TIISSAD

Chapitre 3

Spécifications fonctionnelles

3.1 Problématique

Cette section décrit le découpage, et le jalonnement des tâches du projet. Nous commencerons par un état des lieux des éléments existants puis nous décrirons une décomposition du projet afin d'aboutir à un planning. Pour clore ce chapitre une section sera réservé aux tâches pouvant représenter un écueil au projet. Mais tout d'abord définissons les objectifs du projet et le travail à produire.

Objectif : L'objectif majeur est que le robot apprenne par lui-même des comportements «intelligents» en utilisant le formalisme de l'apprentissage par renforcement.

3.1.1 Décomposition du projet

Le projet peut être découpé en quatre phases majeures qui sont :

1. Étudier le formalisme d'apprentissage par renforcement et notamment sur les robots ;
2. Prendre en main et comprendre l'utilisation des robots Wifibots ;
3. Tester l'algorithme mis en place au cours d'un stage précédent ;
4. Étudier la possibilité de faire converger l'algorithme plus rapidemment ;

Étude du formalisme Plusieurs aspects sont à considérés dans ce formalisme. Il y a d'une part comprendre le formalise général de l'apprentissage par renforcement (Chapitre 4) et comprendre les problématiques liées à la mise en place de telle solution sur des robots.

Prise en main du robot Le wifibot dispose déjà d'une librairie permettant de réaliser la plupart des fonctions désirées. Dans un souci d'homogénéité avec les futurs technologies que l'on va utiliser (URBI) une nouvelle librairie doit être développée.

Test des algorithmes utilisés Implémentation et tests des algorithmes précédemments utilisés. Utilisation d'une interface de simulation mettant à l'épreuve l'algorithme.

Amélioration et optimisations des algorithmes Utilisation des traces d'éligibilité (section 4.4.3) pour converger plus rapidement. Amélioration de l'algorithme utilisé et présenté en chapitre 5.

3.2 Macro Planning

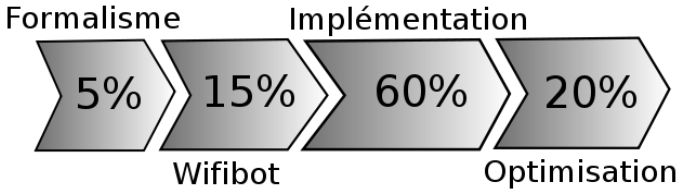


FIG. 3.1 – Vue globale de la répartition des tâches

3.3 Fonctionnalités

Phase 1 La première partie de cette phase est la bibliographie autour de l'apprentissage par renforcement. Il s'agit de comprendre les algorithmes liés à cette discipline. L'implémentation de quelques principes de bases semblent tout indiquée. Des limitations «a priori» vont apparaître concernant l'apprentissage par renforcement dans un environnement continu, avec peu d'exemples (pour que ce soit plus rapide).

Phase 2 La seconde phase concerne l'utilisation du robot, le wifibot avec une librairie déjà mise en place. Une nouvelle spécification doit être conçue pour répondre aux nouvelles attentes d'homogénéité. Des contraintes d'apprentissage vont apparaître liés à la lenteur de l'apprentissage (chaque pas d'apprentissage nécessite un mouvement du robot). On devine également des problèmes de fiabilités des robots, des imprécisions.

Phrase 3 Cette phase consiste à recoder, optimiser, réutiliser le code mise en place avec pour objectif majeur de converger le plus rapidement possible. L'application finale doit être capable de proposer à l'utilisateur une interface lui permettant de faire apprendre le robot. De plus à partir de cette interface l'utilisateur doit être dans la capacité d'exploiter ce que le robot a appris.

Phase 4 Cette dernière phase concerne l'optimisation et la documentation. Cette phase va donc s'occuper de :
- Vérifier l'algorithme ;
- Intégrer un algorithme de traces d'éligibilité ;
- Améliorer la détection de la cible ;

3.4 Choix des outils

Environnement de développement Les algorithmes doivent être rapides et modulaires. Nous avons donc choisi le C++. L'environnement de développement est xemacs sous linux (ubuntu). De nombreuses librairies ont été utilisées tel que :

- la librairie GSL : Gnu Scientific Library est une bibliothèque écrit en C fournissant des outils de calculs numériques en mathématiques appliqués.
- la librarie libjpeg : librairie open source permettant de manipuler le format jpeg.
- GTK+ : Librairie graphique utilisant le paradigme de la conception objet.
- GTKMM : Wrapper C++ de la librairie GTK+. GTKMM pour moins moins (opposé à C++).
- librairie OpenGL : Open Graphics Library, API multi-plateforme pour la conception d'applications générant des images 3D.

Documentation La documentation sera en deux parties. Une première documentation sera générée à partir des commentaires à l'aide de l'outil Doxygen. Une seconde documentation plus traditionnelle sera jointe pour expliquer les architectures logicielles mises en place.

Gestion des versions Pour permettre à différents acteurs d'intervenir dans le code source nous utilisons la forge subversion de l'inria (gforge.inria.fr).

3.5 Planification

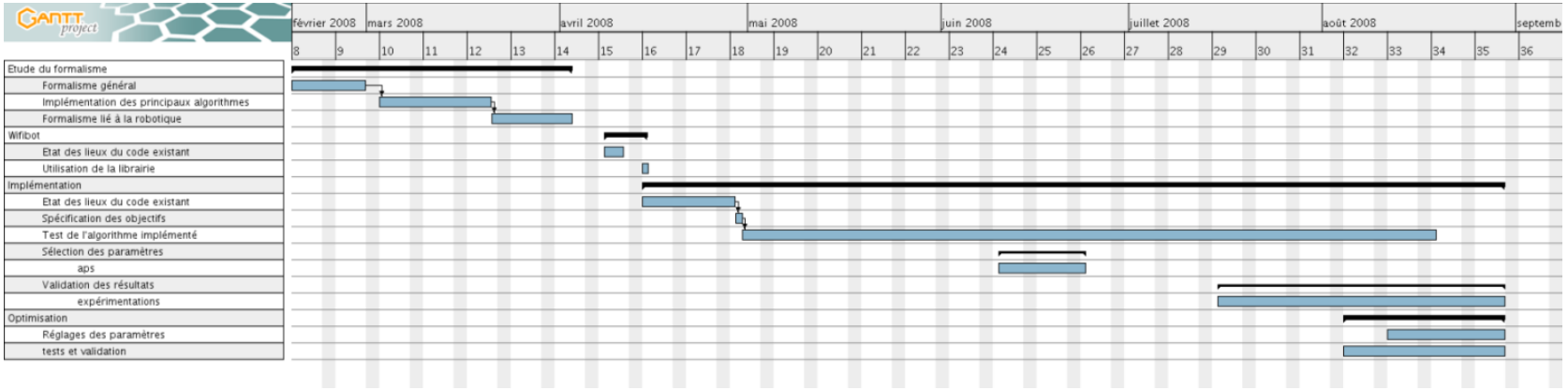


FIG. 3.2 – Planning prévisionnel

3.6 Points du projet à surveiller

Nous allons ici lister les points de difficultés que l'on pourrait rencontrer lors du développement :

- Wifibot : Fiabilité du robot (imprécisions du mouvement, temps de réponse) ;
- Image : Robustesse de l'algorithme de recherche de la cible dans l'image (luminosité, latence de l'image) ;
- Algorithme utilisé : Efficacité de la régression ;
- Temps d'apprentissage : chaque mouvement du robot devrait durer quelques secondes, la durée d'un apprentissage sera donc peut être conséquent.

3.7 Tests

Deux mécanismes de test doivent être prévus. Le premier doit permettre d'éprouver l'implémentation du modèle d'algorithme ; à l'aide par exemple d'un simulateur. Le second mécanisme doit permettre de tester l'interface logicielle proposée à l'utilisateur.

Chapitre 4

Principes généraux de l'apprentissage par renforcement

Dans ce chapitre nous allons décrire le cadre formel de l'apprentissage par renforcement. Ce cadre formel est basé sur les principes des Processus Décisionnels de Markov [**?**] que nous allons étudier en tout premier. Puis nous étudierons les algorithmes d'apprentissage par renforcement, les algorithmes de planification ainsi que les algorithmes de traces d'éligibilité. Ce chapitre ne couvre pas l'ensemble du cadre formel mais contient les bases théoriques nécessaires au projet.

4.1 La genèse de l'apprentissage par renforcement

La figure 6.2 présente un agent et son environnement. Un agent est une entité capable d'interagir avec son environnement au travers de perceptions et d'actions. Ces perceptions lui permettrent de connaître son état. Les actions qui sont réalisées sur ces états peuvent aboutir à l'obtention de pénalités ou de récompenses. L'agent doit réaliser une succession d'actions lui permettant d'atteindre un objectif donné. Cet objectif est en général de maximiser un critère (comme le gain) sur le long terme.

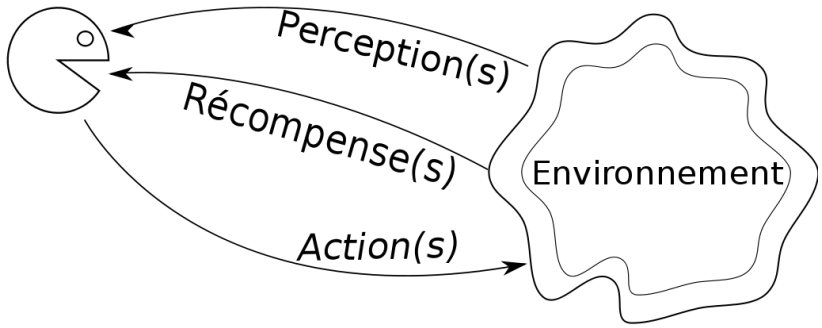


FIG. 4.1 – Boucle en apprentissage par renforcement : un agent **perçoit** un environnement et peut réaliser des **actions** avec lesquelles ils récupèrent des **récompenses**.

Il n'y a pas de définition unanime de **l'agent**, cependant nous pouvons retenir celle-ci : « un agent est une entité capable d'interagir avec son environnement. Doté d'intelligence, il est capable à l'aide de ces interactions d'évoluer selon des critères précis. Ces interactions sont composées d'actions de la part de l'agent et d'un système de récompenses (positive et négative) de la part de l'environnement». (Ferber, 1995).

En ce qui concerne **l'environnement**, on peut le définir comme une extension des chaînes de Markov, nommé Processus de Décisions Markoviens (ou PDM). Les chaînes de Markov permettent d'étudier l'évolution des états d'un système en connaissant les lois qui régissent les transitions (la probabilité de passer d'un état à un autre). Les

PDM ajoutent à ceux-ci la théorie de l'utilité et de la décision (les actions). Dans la section suivante nous allons définir plus précisément ces notions.

4.2 Processus Décisionnels de Markov

Les processus décisionnels de Markov (ou *PDM*) sont des processus stochastiques contrôlés possédant la propriété Markovienne, durant lesquels on obtient des récompenses lors des transitions et permettant de transiter d'état en état. **Un processus stochastique** est un processus qui a une évolution aléatoire dans le temps. **La propriété de Markov** signifie que dans un tel processus, la distribution conditionnelle de probabilité dans des états futurs ne dépend que de l'état présent et pas du passé.

Considérons un système où l'on peut transiter d'un état à un autre à l'aide d'actions. Pour chacune de ces actions une distribution de probabilité est associée ce qui signifie qu'une action n'aura pas forcement le même résultat. Un PDM est un modèle théorique permettant de contrôler un système évaluant de façon stochastique. Etudions la figure 4.2 celle-ci est composée de trois états (noté $S0$, $S1$, $S2$) et deux actions ($a0$, $a1$). Un **état**, s_t, est le résultat de la perception de l'agent au temps t ($s_t \in S$). Une **action**, a, permet à l'agent de modifier son environnement. Pour chaque état, on dispose d'actions dont les effets sont stochastiques. En d'autres termes, à chaque action émise, est associée une probabilité non nulle de transiter vers un autre état et d'obtenir une possible récompense. Par exemple lorsque l'on est à l'état $S2$ et que l'on réalise l'action $a0$ on dispose d'une probabilité de 20% de rester en S_2 et 80% d'aller en S_0.

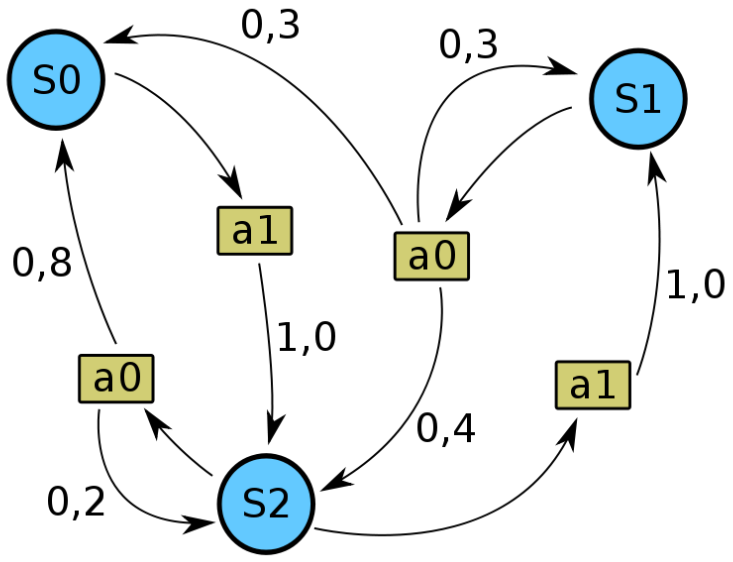


FIG. 4.2 – PDM composé de 3 états et de 2 actions.

A chaque pas de temps t, la probabilité de passer de s à s' en faisant une action a dépend seulement de s et de a et pas des états précédents :

$$P(s_{n+1} = s^{'}|s_n) = P(s_{n+1} = s^{'}|s_n, s_{n-1}, s_{n-2}, ...s_1, s_0) \tag{4.1}$$

4.3 Définition formelle

Les processus de décisions Markoviens sont caractérisés par le tuple **<S,A,T,R>**.

- **S** est l'ensemble fini d'états. Il contient toutes les configurations susceptibles d'être atteintes par un agent dans son environnement.
- **A** est l'ensemble fini d'actions. Il contient toutes les actions qu'un agent pourrait être amené à mettre en œuvre pour explorer son environnement.
- **T** est une fonction de transition qui pour tout couple *état-action* fait correspondre une distribution de probabilité d'arrivée dans un état donné. En d'autre terme, $T : S * A * S \rightarrow [0, 1]$. Ce paramètre permet de caractériser la dynamique du système, car il offre une loi d'évolution d'un agent dans son environnement.
- **R** est une fonction de récompense qui pour tout couple *état-action* fait correspondre une récompense réelle. En d'autre terme, $R : S * AR$. Cette fonction caractérise la motivation de l'agent.

4.3.1 Politique

A chaque pas de temps t, l'agent perçoit son état $s_t \in S$ et l'ensemble des actions possibles depuis cette état $A(s_t)$. Il choisi une action $a \in A$ selon une politique π tel que $a_t = \pi(s_t)$ et reçoit de l'environnement un nouvel état s_{t+1} et une récompense associée r. Un politique est donc une fonction que l'on nomme $\pi : S \longrightarrow A$.

4.3.2 Fonction valeur

La **fonction Q** (qualité) d'un état s et d'une action a sous une politique π est le retour moyen quand on exécute l'action a depuis s, puis que l'on suit π :

$$Q^{\pi}(s, a) = r(s, a) + \gamma \sum_{s' \in S} \mathcal{P}^{a}_{ss'} V^{\pi}(s') \tag{4.2}$$

On connaît Q^{π} une fois que l'on a évalué la fonction valeur V^{π}. La fonction valeur d'un état s sous une politique π, noté $V^{\pi}(s)$ est le retour moyen obtenu quand on applique π au départ de s. En adoptant une politique π dans un état s_t, cette fonction valeur retourne l'espérance de gain :

$$V\ (s) = E\ [\ _{t=0}\ ^{t} r_t | s_t = s] \tag{4.3}$$

Il existe une fonction Q optimale avec :

$$Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a) \tag{4.4}$$

On peut écrire l'équation de V^{*} en fonction de Q^{*} :

$$V^{*}(s) = \max_{a \in A} Q^{*}(s, a) \tag{4.5}$$

Lorsque l'on connait Q^{*} on peut construire une politique optimale π^{*} tel que :

$$\pi^{*}(s) = \operatorname*{argmax}_{a \in A} Q^{*}(s, a) \tag{4.6}$$

Notons que pour tous les $MDP < S, A, T, R >$ il existe un politique optimale notée $\pi*$ qui maximise l'espérance de gain pour chaque état tel que :

$$\forall s \in S, \forall \pi, V^{\pi^*} \geq V^{\pi} \tag{4.7}$$

4.3.3 Equation de Bellman

L'équation de Bellman donne une relation récursive sur $V^{\pi}(s)$ pour un état s en la liant à la fonction de valeur de ses états successeurs.

$$\forall s, V(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V(s') \tag{4.8}$$

- $r(s, \pi(s))$ est la fonction récompense associée de l'action $\pi(s)$ dans l'état s.
- $T(s, \pi(s), s')$ est la probabilité de transition de l'état s à l'état s' en adoptant la politique $\pi(s)$.

4.4 Les algorithmes

4.4.1 Planification

Dans ce type d'algorithme, l'agent dispose d'un modèle du monde (<S,A,T,R>).

Policy Evaluation

Avant de planifier une politique il est nécessaire de l'évaluer. On utilise pour cela un algorithme nommé *policy evaluation* qui permet d'évaluer pour une politique π donné la *fonction valeur* associée :

Algorithme 4.1 : Policy Evaluation

Entrées :
π, la politique à évaluer
ϵ
Sorties :
V_t
début
  Initialiser $V_0(s) = 0$, pour tous les $s \in S$
  tant que $max_s|V_t(s) - V_{t-1}(s)| < \epsilon$ **faire**
    $t \leftarrow t + 1$
    pour *tous* $s \in S$ **faire**
      $V_t(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_{t-1}(s')$
    fin
  fin
  retourner V_t
fin

Policy Iteration

Policy Iteration est un algorithme itératif. Pour chacune de ces itérations, on cherche une politique que l'on va appeler π^+, meilleure que la politique courante. De tel sorte que l'on tende vers les politique optimale π^*.

$$\pi^+(s) = \underset{a \in A}{\operatorname{argmax}} Q^\pi(s,a) \tag{4.9}$$

$$\pi^+(s) = \underset{a \in A}{\operatorname{argmax}} [R(s,a) + \gamma \sum_{s'} p(s'|s,a) V\pi(s')] \tag{4.10}$$

Cette nouvelle politique est meilleure que la précédente, on a en effet $\forall s V_{\pi^+}(s) \geq V_\pi(s)$. **Policy Iteration** permet d'obtenir la politique optimale $\pi*$ car elle satisfait les équations d'optimalité de Bellman. L'algorithme converge en un nombre fini d'étapes ; mais il peut être coûteux en temps puisque l'on évalue π à chaque itération.

Algorithme 4.2 : Policy Iteration

Entrées :
$< S, A, T, R >$
π
Sorties :
$\pi*$
début
  $\pi, \pi^+ \leftarrow$ une politique aléatoire
  tant que $\pi^+ <> \pi$ **faire**
    $Q^\pi \longleftarrow$ Evaluation de la politique π
    Construire une nouvelle politique : $\pi^+(s) = \underset{a \in A}{\operatorname{argmax}} Q^\pi(s,a)$, pour tout $s \in S$.
    $\pi \longleftarrow \pi^+$
  fin
fin

Value Iteration

Value Iteration consiste à résoudre l'équation de Bellman en $V^*(s)$ par une méthode itérative de type point fixe. On fixe V_0, puis de manière incrémentale on génère une suite V_1, V_2, V_3, ... de fonctions valeurs qui convergent V^* (équation 4.11) :

$$\forall s V_{n+1}(s) = \max_a [r(s,a) + \gamma \sum_{s'} p(s'|s,a) V_n(s')] \tag{4.11}$$

L'algorithme 3 présente une façon d'écrire value iteration. Le critère d'arrêt de cet algorithme dépend de la précision que l'on souhaite, c'est-à-dire de l'approximation du calcul de V_t qui se rapproche de V^*. Cette précision est donnée par l'équation 4.12. Pour $\epsilon > 0$ on stoppe l'algorithme lorsque $max_{s \in S} |V_t(s) - V_{t-1}(s)| < \epsilon$, on a alors :

$$\max_{s \in S} |V^*(s) - V_t(s)| \leq \frac{\gamma}{1-\gamma} * \underbrace{\max_{s \in S} |V_t(s) - V_{t-1}(s)|}_{\epsilon} \tag{4.12}$$

Algorithme 4.3 : Value Iteration

Entrées :
$< S, A, T, R >$
Sorties :
Approximation de V^*
début
  Initialiser V_0, pour tous les $s \in S$
  Initialiser $t = 0$
  tant que $max_{s \in S} |V_t(s) - V_{t-1}(s)| < \epsilon$ **faire**
    pour *tout* $s \in S$ **faire**
      $V_{t+1}(s) \longleftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s') * [r(s, a) + \gamma * V_t(s')]$
    fin
  fin
fin

4.4.2 Q-learning

Contrairement aux algorithmes de planifications qui supposent une connaissance du modèle de l'environnement, le Q-Learning est un algorithme d'apprentissage par renforcement qui associe des Q-valeurs à chaque couple état-action, noté $Q(s, a)$. Ne disposant pas de modèle du système, à chaque état, l'agent va émettre de façon aléatoire une action, puis observer le retour de son environnement. La Q-valeur d'un couple état (s) action (a) s'exprime de la façon suivante :

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^\pi(s') \tag{4.13}$$

L'apprentissage par renforcement calcule donc la valeur d'un état associé à une action. Cette Q-valeur est composée à la fois de la nouvelle interaction mais prend en compte également le passé. En réitérant plusieurs fois les actions aux différents états on **renforce** la Q-valeur. L'équation 4.14 indique la façon dont est mis à jour le Q-learning, de plus l'algorithme 4 présente une façon de l'écrire.

$$Q(s, a) \leftarrow \underbrace{(1 - \alpha) Q(s, a)}_{expériences - précédentes} + \underbrace{\alpha (r + \gamma max_{a' \in A} Q(s', a'))}_{nouvelle - expérience} \tag{4.14}$$

Le paramètre α désigne le taux d'apprentissage du système. Si α est près de 1 alors l'agent privilégiera la nouvelle expérience, si α est près de 0 l'agent privilégiera les expériences passées. On peut imaginer faire varier ce facteur selon que l'on est dans une phase d'exploration ou d'exploitation (en fonction de conditions précises du Q-learning).

Le paramètre γ est le paramètre dit de pondération ou facteur de décompte ($\gamma \in [0, 1[$). Il permet de régler l'importance que l'on donne aux récompenses futures par rapport aux récompenses immédiates. Si γ est près de 1 alors les récompenses futures sont prises en compte, à l'inverse près de 0 les récompenses futures sont négligeables. Si γ vaut 0, cela revient à dire que l'on prend en compte un horizon d'un coup. Plus γ se rapproche de 1 et plus l'horizon est important.

Algorithme 4.4 : Q-Learning

Entrées :
$N \longleftarrow$ nombre d'itérations
Sorties :
Q_N
début
  Initialisation de Q_0
  pour $n \in N$ **faire**
    $s_n \leftarrow$ choix d'un état
    $a_n \leftarrow$ choix d'une action
    $(s_n^{'}, r_n) \leftarrow Simuler(s_n, a_n)$
    $Q_{n+1} \leftarrow Q_n$
    $Q_{n+1}(s_n, a_n) \leftarrow Q_n(s_n, a_n) + \alpha_n(s_n, a_n) r_n + \gamma max_b Q_n(s_n^{'}, b) - Q_n(s_n, a_n)$
  fin
fin

4.4.3 Traces d'éligibilité

Les traces d'éligibilité permettent de propager une q-valeur calculée sur les états précédents. Une trace d'éligibilité est associée à chaque paire état-action. Elle est augmentée à chaque évaluation de la paire état-action, sinon à chaque pas de temps elle décroît.

Afin de converger plus vite, certains algorithmes utilisent une mémoire de transition qu'ils ont effectués lors d'une expérience afin d'en propager au maximum les effets. Cette mémoire est appelée **traces d'éligibilité**.

Lorsque l'on utilise des algorithmes tel que le $Q-learning$, on calcule la Q-valeur de chaque paire état-action. Les traces d'éligibilité permettent de propager ce calcul afin d'en maximiser la portée. Il existe plusieurs façon de mettre en place les traces d'éligibilité. On peut utiliser les différences temporelles, les traces d'éligibilité avec réinitialisation, les traces d'éligibilité accumulatives. Ce sont ces dernières que nous allons expliquer.

Les traces d'éligibilité utilisent un facteur de pondération noté $\lambda \in [0, 1]$. Dans le cas où $\lambda = 0$, la sauvegarde sera constitué de la dernière composante ; si $\lambda = 1$ alors tout l'historique sera concerné par la sauvegarde, tel que :

$$R_t^{\gamma} = (1 - \gamma) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \tag{4.15}$$

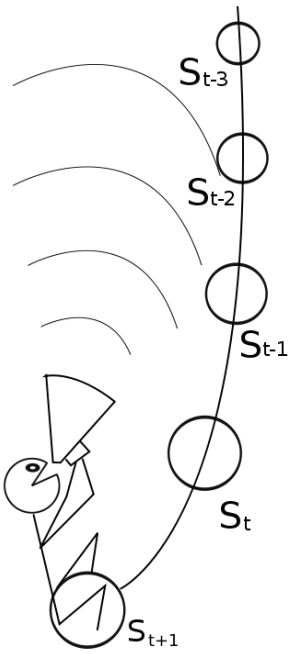


FIG. 4.3 – Propagation des traces

4.5 Références

Voici les références utilisées pour ce chapitre :
[**?**], [**?**], [**?**], [**?**]

Chapitre 5

Spécifications techniques

Dans ce chapitre nous allons écrire le cahier des charges techniques du projet. En s'appuyant sur la thèse du docteur Smart, nous allons décrire une approche possible d'implémentation d'un apprentissage par renforcement sur un robot, dans un environnement continu.

5.1 Contexte de la tâche

La tâche consiste à apprendre au robot à se rapprocher d'une cible. Le principe est de faire de l'apprentissage par renforcement en exploitant du Q-learning. L'apprentissage est composé de deux phases. La première phase permet de créer des points d'entraînements composés d'un état et d'une action et d'en définir une Q-valeur. La seconde phase est de choisir une action en fonction de la meilleure action à réaliser. Les Q-valeurs sont alors interpolées.

L'algorithme utilisé étant du Q-learning, nous pouvons définir le **MDP** (section 5.3). Le robot en question est un **wifibot** dont les caractéristiques sont présentées à l'annexe A. Un description est également faite de la **cible** au chapitre 5.4, page 19.

Les deux supports principaux utilisés pour cette tâche sont la thèse du Dr William Donald Smart [**?**] et l'article de Jing Peng and Ronald J. William [**?**]. Le principe définit dans la thèse du Dr Smart, est de pouvoir interpoler une q-valeur d'un point (distance-angle-action) en fonction des points d'entrainements réalisés par l'utilisateur. L'interpolation est une approximation de la valeur. Pour cela nous allons utiliser une régression. Pour mettre en place cet outil nous avons besoin d'une structure de stockage et d'un principe d'interpolation. C'est ce que nous allons développer dans la suite de ce chapitre.

5.2 Etat des lieux

Mon stage a eu comme point de départ les travaux très avancés de Nicolas Beaufort :

- Pilotage du wifibot effectué ;
- Implémentation du kdtree en C ;
- Implémentation de la thèse du docteur Smart en C ;

5.3 Description de l'environnement

Cette section a pour objectif de définir le contexte d'évolution du robot, c'est à dire de décrire partiellement ce que l'on connaît du MDP. Voici ce que l'on connaît du tuple <S,A,T,R>.

S : état Un état du robot est constitué d'une distance et d'un angle par rapport à la cible. Les états sont continus, c'est à dire qu'il y a en théorie une infinité de valeurs.

Cependant cette notion est relativisée par les capacités techniques du robot qui nous imposent quelques limites. Les intervalles sont définis à la section 5.4. Les états sont normalisés.

A : action Les actions du robot sont stochastiques, c'est à dire qu'une même action pour un même état donnera un résultat aléatoire. Le résultat aléatoire se justifie par l'utilisation du robot puisque deux commandes successives n'auront pas le même comportement. Le robot dispose des actions suivantes :
- Ne rien faire : avancer à une vitesse donnée pendant 2s ;
- Avancer ;
- Reculer ;
- Tourner à gauche ;
- Tourner à droite ;

R : Récompense Une récompense est obtenue par le robot lorsque qu'il est à une distance inférieure à un certain seuil. Les distances étant normalisées nous avons définit le seuil à 80%.

5.4 Traitement de l'image

Pour que le wifibot se repère dans l'espace il utilise en entrée l'image de sa caméra. L'image est récupérée sous la forme d'une JPEG depuis un flux HTTP généré par la caméra elle-même (caméra IP). Ce chapitre va décrire la méthode utilisée pour repérer la cible dans l'image et la façon d'en obtenir les informations qui nous intéressent.

Description de la cible : La cible est composée de trois cercles concentriques de couleurs différentes : rouge, bleu et jaune.

Quelles informations nous sont utiles : Pour réaliser notre apprentissage nous devons être en mesure de localiser le wifibot par rapport à la cible. Nous avons décidédans un souci de simplicité, d'utiliser deux informations : l'**angle** et la **distance**. Nous n'avons pas la capacité d'obtenir ces informations précises dans le contexte d'utilisation du wifibot. Nous allons utiliser donc deux métriques depuis l'image : la distance sera représentée par la largeur de la cible dans l'image, l'angle quand à lui sera représenté par la distance entre le début de l'image et le début de la cible (figure 5.1). Comme indiqué dans le début de ce chapitre un minimum et un maximum sont définis pour les angles et les distances :
- distance $\in [40, 200]$
- angle $\in [0, 600]$

5.4.1 Fonctionnement de la détection

La détection de la cible se déroule en trois phases (une fois l'image acquise par protocole HTTP) :
- conversion de l'image en bitmap ;
- conversion de l'image au format HSL ;
- recherche de la cible ;

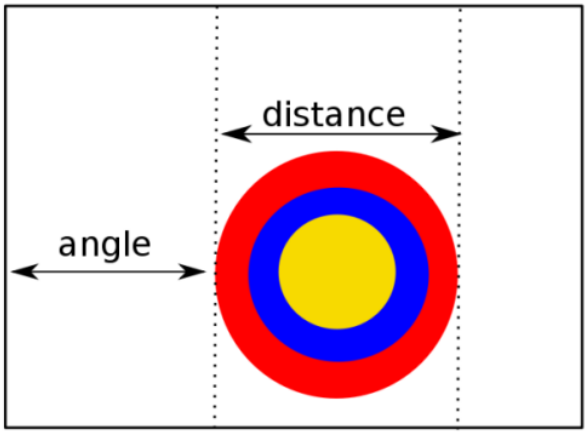


FIG. 5.1 – Informations obtenues depuis l'image

Lorsque l'image est acquise au format jpeg, la classe CImage la convertit en bitmap à l'aide de la librairie «libjpeg». Puis l'opération de conversion en teinte est effectuée. Ce format est couramment appelée **HSL** pour Hue, Saturation, Lightness ou **TSL** en Français pour Teinte Saturation lumière. La figure 5.2 contient un exemple de conversion en HSL.

Le format HSL consiste à coder la couleur selon des critères physiologiques :
- la teinte : Perception de la couleur ;
- la saturation : Pureté de la couleur (vif ou terne) ;
- la luminance : quantité de lumière de la couleur (claire ou sombre).

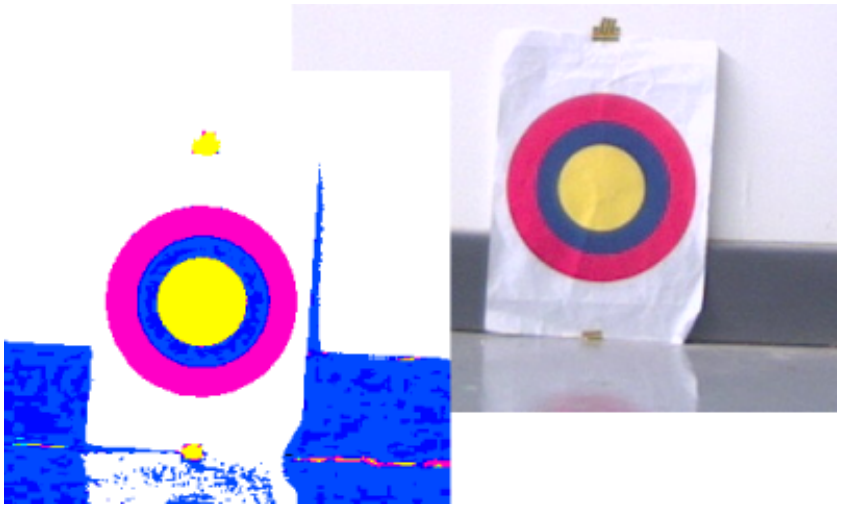


FIG. 5.2 – A droite l'image originelle, à gauche elle est convertie en HSL

Recherche de la cible : Nous allons détailler ici le fonctionnement de l'algorithme de recherche de la cible. Dans un premier temps l'algorithme 12 analyse l'image en entrée

et génère un tableau de liste de séquence de couleurs qui ont pour caractéristiques :
- la couleur concernée ;
- l'indice de pixel de début ;
- l'indice de pixel de fin.

Suite à cette analyse nous recherchons le **motif** suivant : rose, bleu et jaune dont voici l'illustration :

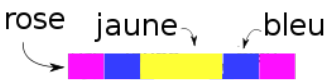


FIG. 5.3 – Motif recherché

Si plusieurs motifs sont trouvés dans l'image, seul le plus gros est pris en compte. Pour qu'un motif soit considéré valide il doit être répliqué sur plusieurs lignes. Le motif doit respecter un certain nombre de critères métriques précis pour être accepté afin d'éviter les «bruits» de l'image :
- La taille de la première séquence rose doit être similaire à la seconde ;
- La fin de la première séquence et le début de la suivante doivent être proche ;
- La fin de la deuxième séquence et le début de la suivant doivent être également proche ;

Vous trouverez en Annexes D l'algorithme 12 qui permet de détecter la cible dans l'image.

5.5 Kd-tree

Le **kd-tree** [**?**] est une structure mémoire optimisée pour le stockage de points dans l'espace. Il est particulièrement adapté pour organiser des données d'entrées dans un espace à k-dimensions selon leur répartition spatiale. Le kd-tree pour «k-dimensions tree» est un cas particulier du Binary Space Partitionning (BSP) [**?**]. Nous allons étudier ses caractéristiques et l'implémentation qui en découle.

5.5.1 Description

Le kd-tree est un arbre binaire. Un arbre est une structure de données hiérarchisée où chaque élément est appelé nœud. Le nœud initial est la **tête** (ou racine). Un arbre **binaire** indique que le niveau inférieur d'un nœud est composé au maximum de deux sous nœuds (que l'on appelle en général gauche et droite). La figure 5.4 illustre ces principes.

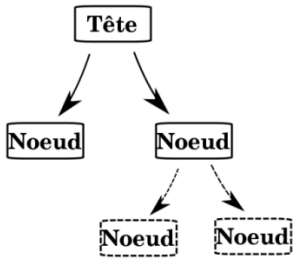


FIG. 5.4 – Arbre binaire

Un kd-tree possède deux caractéristiques qui sont le nombre de dimensions et le le nombre d'entrée par **feuille**. Une **feuille** est un nœud qui n'a pas d'enfant et qui stocke donc des valeurs. Le **nombre de dimensions** revient à spécifier le nombre de valeurs par entrée à stocker par feuille. La **limite du nombre d'entrées** est le paramètre permettant de spécifier la profondeur de l'arbre. Plus cette valeur est petite et plus l'arbre sera profond. Ce paramètre est également très important pour le parcours de l'arbre comme nous le verrons par la suite. Lorsque la quantité d'entrée dépasse le maximum autorisé dans un nœud on doit alors découper ce nœud. Cette subdivision, d'un kdtree, intervient sur la valeur médiane de la structure. Ce stockage permet de simplifier la construction et le parcours de l'arbre. Le kd-tree a donc un rôle double :

- Organiser l'espace pour accélérer le traitement des données ;
- Structurer les données sous forme d'un arbre binaire.

5.5.2 Mise en œuvre

Composition d'un nœud : Chaque nœud du kd-tree contient : un index d'une dimension pour la coupure, une valeur de coupure, un nœud enfant gauche, un nœud enfant droit, un flag pour savoir si c'est une feuille, un système stockage des valeurs de la feuille.

Insertion d'un point : L'algorithme 10 présenté à l'annexe C (page 44) permet de décrire le comportement du kd-tree lorsque l'on ajoute un nouveau point. Dans un premier temps on doit parcourir l'arbre pour trouver la feuille qui nous convient. Pour trouver cette feuille, il faut comparer la valeur du point que l'on souhaite ajouter à la dimension la plus répandue dans le kdtree et la valeur de chaque afin de trouver la bonne branche. Lorsque cette feuille est trouvée on ajoute notre point.

Découpage d'un nœud : Si le nombre de points de cette feuille dépasse la limite fixée, la feuille est découpée en deux et devient un nœud. Les points stockés dans ce nœud sont alors répartis en deux sous feuilles en fonction de leur valeur de dimension

la plus répandue (Algorithme 11, annexe C). Pour mieux comprendre le mécanisme mise en place, nous allons décrire la figure 5.5.

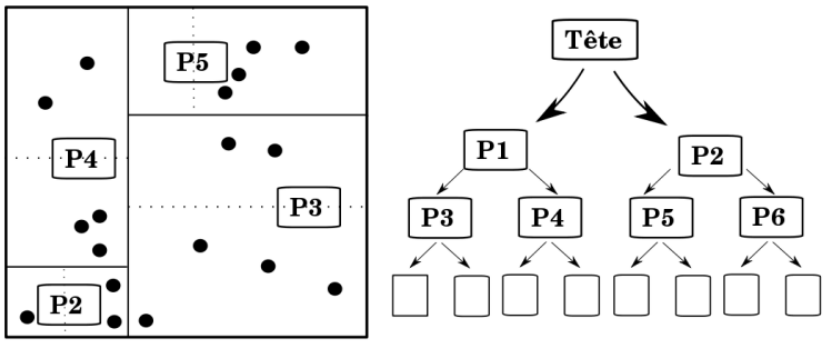


FIG. 5.5 – Deux représentations d'un kd-tree

La partie droite de la figure 5.5 représente la structure mémoire du kd-tree. On voit donc apparaître les points tout en bas sous forme de feuilles de données. A gauche de la figure on peut observer une décomposition du kd-tree plus intuitive (découpage visuelle). Au fur et à mesure de l'ajout de point de nouvelle profondeur vont apparaîstre.

Parcours de l'arbre : Tout l'intérêt de cette structure dans notre application est de rechercher rapidement les voisins d'un point. Le parcours de l'arbre est récursive : chaque itération compare la valeur du point à la valeur du nœud et évolue à droite ou à gauche en fonction de cela.

5.6 Enveloppe convexe

L'enveloppe convexe d'un ensemble de points consiste à trouver le plus petit polygone incluant tous les points initiaux. C'est donc un ensemble convexe de taille minimal qui contient les objets de départ. En mathématique un objet est dit convexe si pour toute paire de point A, B de cet objet, segment $[AB]$ qui les joint est entièrement contenu dans l'objet.

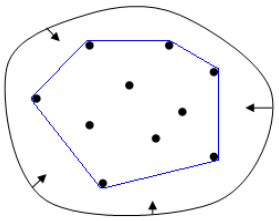


FIG. 5.6 – L'enveloppe convexe est ici en bleue.

5.7 Régression locale pondérée

La régression locale pondérée s'inspire de la régression linéaire simple. Effectuer une régression linéaire d'un caractère à expliquer y sur un caractère explicatif x c'est trouver les coefficients a et b tels que la variance de la série des résidus $y - ax - b$ est minimale. La figure 5.7 présente la «meilleure» droite en fonction du nuage de point. La régression locale pondérée est donc une régression calculée sur un point (x_i, y_i)

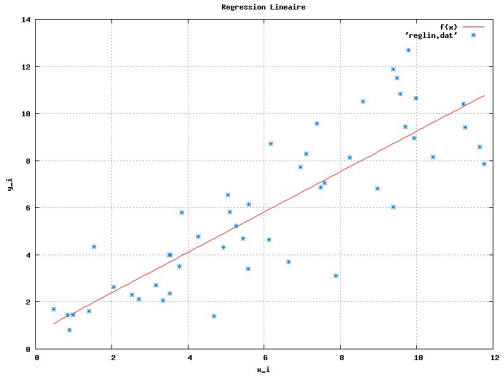


FIG. 5.7 – Régression linéaire simple

précis. Elle est pondérée parce qu'elle utilise un système d'influence en rapport avec sa distance. Les points pour le calcul de la régression auront plus ou moins d'influence en fonction de la distance entre ces points et le point à évaluer. Plus le point faisant partie de la régression est loin et moins il aura de «poids» dans le calcul. Pour effectuer cette régression il faut résoudre ce système :

$$AX = B \tag{5.1}$$

A, B et X sont des matrices. A est construit avec les points d'entraînements pondérés par un poids. B correspond au Q-valeur. X est le résultat de la régression. Pour résoudre X, il suffit d'inverser la matrice A en réalisant $X = A^{-1}B$. Cependant A n'est pas toujours inversible, c'est pourquoi le Dr Smart préconise l'utilisation d'une matrice **pseudo-inverse** [**?**].

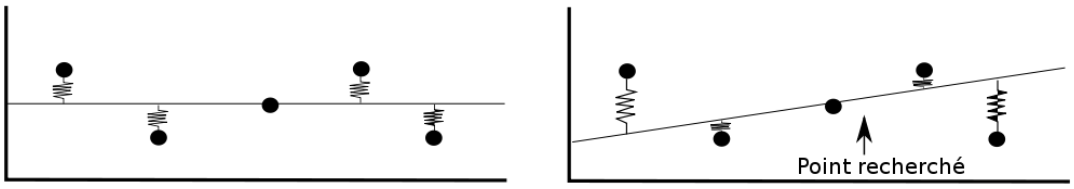


FIG. 5.8 – Régression linéaire simple (à gauche) et régression locale pondérée (à droite)

5.8 L'algorithme Hedger

L'algorithme Hedger est composé de trois blocs principaux qui sont l'apprentissage (Training), la prédiction (Prediction), et la régression locale pondérée (LWR). La figure 5.9 permet de comprendre les interactions de ces trois blocs. Dans la suite de section, nous allons décrire la fonction de chacun de ces blocs.

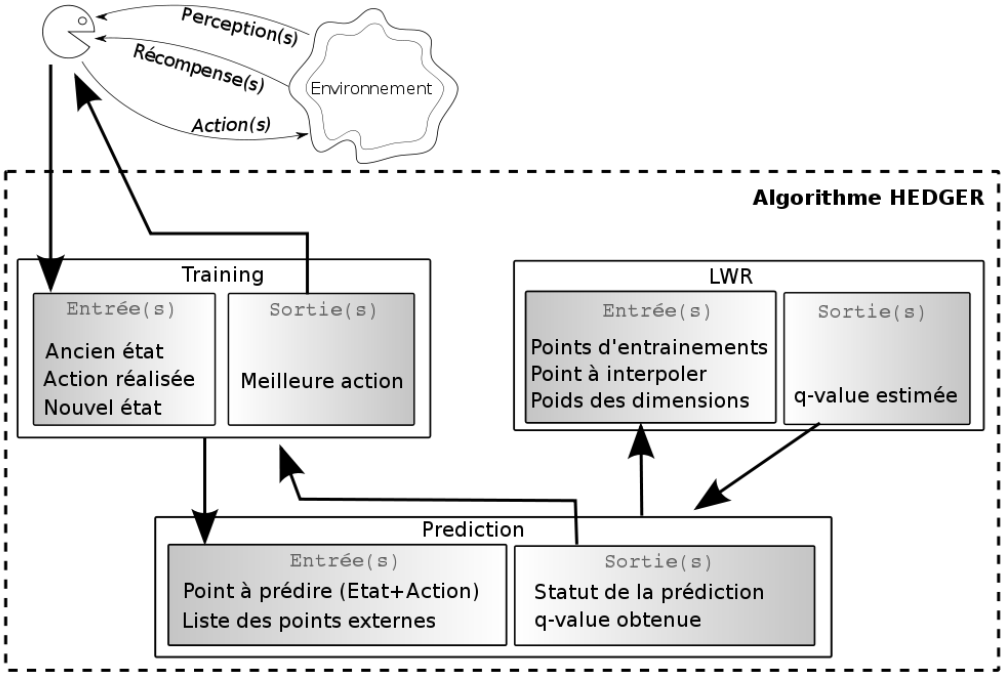


FIG. 5.9 – Le fonctionnement de l'algorithme HEDGER

5.8.1 Bloc Training

L'algorithme 6 à l'annexe B présente le fonctionnement de ce bloc.

Pour commencer, le robot est dans un état s_1. L'utilisateur lui indique de réaliser l'action a_1 et le robot se retrouve alors dans l'état s_2. En utilisant le premier bloc *Training* le robot peut estimer la prochaine meilleure action. Pour réaliser cette estimation *Training* doit avoir en entrée l'état précédent, l'action réalisée et l'état obtenu.

Grossièrement Training a pour rôle de mettre à jour les q-valeurs et d'en ajouter. C'est cette fonction qui réalise l'apprentissage. Elle fonctionne comme du Q-learning classique (voir chapitre 4, section 4.4.2).

Dans notre exemple Training calcule la nouvelle q-valeur tel que :

$$newQvalue \leftarrow \alpha(r + \gamma Q_{s_2} - Q_{(s_1,a_1)}) + Q_{(s_1,a_1)} \tag{5.2}$$

Q_{s_2} est la meilleure prédiction de q-valeur depuis l'état s_2. $Q_{(s_1,a_1)}$ est la q-valeur prédite la paire état-action : s_1, a_1. Si le point calculé est déjà dans l'arbre on modifie sa valeur ainsi que la valeur de ses voisins. Dans le cas inverse le point est ajouté à l'arbre.

5.8.2 Bloc Prediction

L'algorithme 7 à l'annexe B présente le fonctionnement de ce bloc.

Le bloc prédiction a pour fonction de prédire la q-valeur d'un point. Si le point exacte existe dans l'arbre sa q-valeur est renvoyée. Sinon la fonction collecte k points les plus proches. Si le nombre de point est suffisant on calcule si le point que l'on recherche est dans l'enveloppe convexe. Cette enveloppe convexe est appelé IVH (*independent variable hull*). Pour vérifier si un point $\vec{x}$ est dans l'IVH on se base sur la liste des points d'entraînements. On crée une matrice X possédant ces points d'entraînements et on calcule la **hat matrix [?]** comme suit :

$$V = X(X^T X)^{-1} X^T \tag{5.3}$$

Puis pour savoir si le point $\vec{x}$ est dans l'IVH formé par les points de X :

$$\underbrace{\vec{x}^T (X^T X)^{-1} \vec{x}}_{H} \leq \max_i v_{ii} \tag{5.4}$$

v_{ii} représente la diagonale de la matrice V. $\max_i v_{ii}$ est donc la valeur maximale de cette diagonale. Cette quantité représente un poids de mesure de la distance du point recherché au centre de gravité des points d'entraînements.

Exemple de calcul Nous allons mettre en pratique les notions évoquées ci-dessus avec une liste de quelque points réalisés par le robot :

Distance précédente	Angle précédent	Action	Distance	Angle	Récompense
0,1	0,10	1	0,300	0,20	2
0,3	0,20	1	0,15	0,30	2
0,15	0,30	1	0,20	0,5	2
0,20	0,5	1	0,405	0,40	2

Après avoir utilisé la fonction Training autant de fois que de point on obtient le kd-tree suivant :

Distance	Angle	Action	Q-valeur
0,1	0,1	1	1,8
0,3	0,2	1	1,8
0,15	0,3	1	1,8
0,2	0,5	1	1,8

La matrice X est composé des valeurs du kd-tree, la matrice V est le résultat du calcul définit en 5.3, ici $\max_i v_{ii} = 1.0$.

$$X = \begin{pmatrix} 0,1 & 0,1 & 1 \\ 0,3 & 0,2 & 1 \\ 0,15 & 0,3 & 1 \\ 0,2 & 0,5 & 1 \end{pmatrix} V = \begin{pmatrix} 8.33e-01 & 2.48e-15 & 3.33e-01 & -1.66e-01 \\ 3.49e-15 & 1.00e+00 & 2.12e-15 & 8.73e-16 \\ 3.33e-01 & 1.20e-15 & 3.33e-01 & 3.33e-01 \\ -1.66e-01 & 7.31e-17 & 3.33e-01 & 8.33e-01 \end{pmatrix} \tag{5.5}$$

En appliquant l'équation 5.4 pour le point $\vec{x}_0 = [0.25, 0.25, 1.0]$ on obtient $H[0][0] = 0.45$ ce qui signifie que le point **est** dans la zone IVH, car $\max_i v_{ii} = 1.0$.

En appliquant l'équation 5.4 pour le point $\vec{x}_1 = [0.5, 0.5, 1.0]$ on obtient $H[0][0] = 4.83$ ce qui signifie que le point **n'est pas** dans la zone IVH, car $\max_i v_{ii} = 1.0$.

Le script python utilisé pour cette exemple est disponible à l'annexe E.
La fonction Prediction peut renvoyer différents **statuts** que nous allons décrire :
- Point externe : Ce statut signifie que le point à prédire fait parti de la liste des points externes. Il existe deux types de points externes. Lorsque le robot finit une action et qu'il ne voit plus la cible il n'est plus en mesure de définir une distance et un angle, c'est pourquoi on affecte -1 à ces deux valeurs. D'autres part lorsque le robot initie un apprentissage il ne connaît pas non plus l'emplacement de la cible, dans ce cas on lui affecte les valeurs -2. Ces deux cas multiplié par les actions possibles constituent la liste des points externes. De cette façon ces points n'influent pas sur la régression.
- Point exacte : Cela signifie que le point que l'on cherche à interpoler est déjà renseigné dans la structure de données.
- Pas assez de points : Ce statut indique qu'il n'y pas assez de points d'entraînements pour réaliser l'interpolation, ce paramètre est fixé dans l'algorithme HEDGER.
- En dehors de la zone IVH : Comme l'a illustré notre précédent exemple, cela signifie que le point que l'on cherche à interpoler n'est pas présent dans l'enveloppe convexe.
- En dehors des bornes : Lorsque que l'on est dans la zone IVH, on réalise la régression linéaire pondérée qui retourne la q-valeur estimée. Cette valeur doit être dans la fourchette suivante : $\frac{rmin}{1-\gamma} < Qvaleur < \frac{rmax}{1-\gamma}$, où $rmin$ est la récompense minimum reçue et $rmax$ la maximum. Sinon ce statut est renvoyé.
- Prédiction normale : Prédiction réalisée avec succès.

5.8.3 Bloc LWR

Le bloc LWR est la régression locale pondérée ou (Locally Weigth Regression). Elle permet d'estimer la q-valeur d'un point en fonction d'une liste de points passer en paramètre.

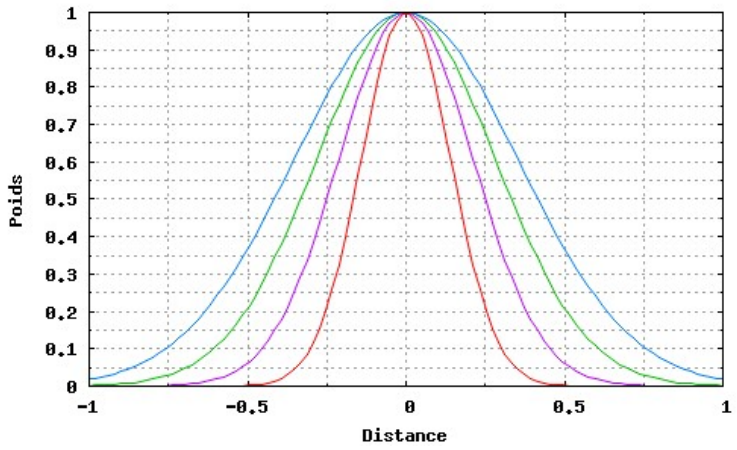


FIG. 5.10 – Exemple de plusieurs bandes passantes

Le principe de cette régression est expliqué à l'algorithme 9 à l'annexe B. Pour réaliser cette régression, les valeurs des points d'entraînements sont multipliées par

un poids que l'on définit pour chaque point comme :

$$e^{-(\frac{d}{h})^2} \tag{5.6}$$

Dans l'équation 5.6, d représente la distance entre le point que l'on cherche à interpoler et le point dont on cherche à calculer le poids. h correspond à la bande passante qui un paramètre de l'algorithme HEDGER. Ce paramètre permet de définir l'influence des points en fonction de la distance. La figure 5.10 présente différentes bandes passantes en fonction de la distance.

5.8.4 Les paramètres

Dans cet algorithme les paramètres sont très importants. Ils doivent fixer en fonction des points d'entraînements. Ainsi avec peu de points d'entraînements il faudra définir peu de points pour permettre une interpolation. A l'inverse avec un nombre de points conséquent il parait important de fixer un nombre de points suffisant. Plus le nombre de points requis pour l'interpolation est important et plus le temps de calcul sera long. On ne peut donc pas ne pas définir de limite quantitative quand au nombre points. Voici une liste des paramètres :

- bande passante (H) ;
- nombre de points minimum pour une interpolation ($points\_min$) ;
- nombre de points maximum pour une interpolation ($points\_max$) ;
- alpha (α) ;
- gamma (γ) ;
- lambda (λ) ;
- seuil ($threshold$) ;

5.9 Algorithme Incremental Multi-Step Q-Learning

Pour converger plus rapidement nous avons utilisé une implémentation des traces d'éligibilité (abordé page 17, section 4.4.3). L'article utilisé «Incremental Multi-Step Q-Learning» [**?**]. Cet article décrit une approche combinée de Q-learning et de $TD(\lambda)$. Ce mélange doit accélérer la méthode d'apprentissage par renforcement.

L'idée est de renforcer l'historique des q-valeurs lorsque l'on obtient une récompense. A chaque pas d'apprentissage on calcule une éligibilité avec la formule suivante :

$$e_t = r_t + \lambda \hat{V}_t(x_{t+1}) - \hat{Q}_t(x_t, a_t) \tag{5.7}$$

- λ est le paramètre de pondération des traces, plus il est proche de 1 et plus l'influence des traces sera importante ;
- $\hat{V}_t(x_{t+1}) \leftarrow argmax_a \hat{Q}_t(x, a)$;
- $\hat{Q}_t(x_t, a_t)$ correspond à la nouvelle q-value ;

Puis on applique cette trace sur l'historique des points stockés préalablement, et on incrémente la valeur de la trace de 1. L'algorithme suivant présente ce procédé :

Algorithme 5.1 : L'algorithme Peng

Entrées :
Le point $\vec{x}$ de départ (composé d'un état x, et d'une action a)
Le point $\vec{xp}$ d'arrivé (composé d'un état x, et d'une action a)
La récompense r obtenue
Sorties :
Mise à jour des Q-valeurs
début
  $\hat{V}_{\vec{xp}} \leftarrow$ Meilleure Q-value de $\vec{xp}$
  $\hat{V}_{\vec{x}} \leftarrow$ Meilleure Q-value de $\vec{x}$
  $e = r + \lambda \hat{V}_{\vec{xp}} - \hat{V}_{\vec{x}} \leftarrow$
  pour *toutes les pairs état-action* (x, a) **faire**
    $Tr(x, a) = \gamma\lambda Tr(x, a)$
    $\hat{Q}_{t+1}(x, a) = \hat{Q}_t(x, a) + \alpha Tr(x, a)e$
  fin
  $Tr(\vec{x}) \leftarrow Tr(\vec{x}) + 1$
fin

Chapitre 6

Gestion du projet APRAM

6.1 Implémentation

6.1.1 Formalisme

Après avoir lu un peu de la bibliographie sur le sujet, mon encadrant m'a conseillé d'implémenter quelques uns des algorithmes de bases du domaine. C'est ainsi que j'ai écrit une petite application en java «MDP_Rat» qui utilise les algorithmes de Q-learning canonique, Q-learning blotzmanien, Q-learning ϵ-greedy, value iteration et policy iteration. La figure 6.1 présente une impression écran du logiciel.

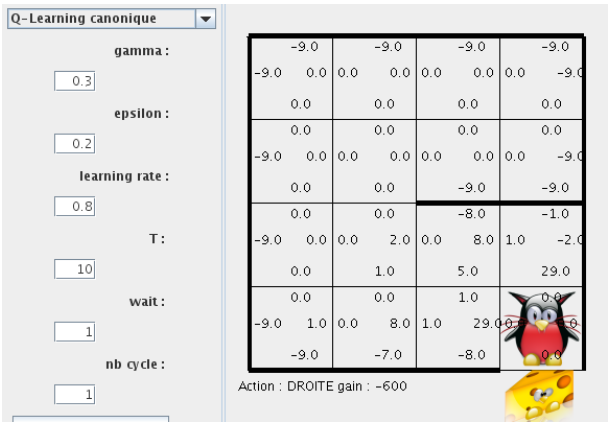


FIG. 6.1 – Application Java : MDP_Rat

6.1.2 Wifibot

caméra Utilisation des pans (mouvements à droite et à gauche), tils (mouvements en haut et en bas), récupération du flux streaming jpeg, utilisation des algorithmes de détection de surface.

matériel Utilisation des commandes I2c pour les roues, les capteurs infrarouges le niveau de batterie, l'activation ou non de l'asservissement.

6.1.3 Test des algorithmes

Tester sur PC Implémenter un logiciel de simulation sur PC pour comprendre les mécanismes mis en place par l'apprentissage.

L'implémentation du modèle proposé par le Dr Smart préconise quelques briques de base tel qu'une classe KD-TREE qui gère ce type de structure, d'une classe Hedger qui contient l'algorithme du même nom, une classe Peng qui met en place du Q-Learning(λ). Ce découpage en classe a été inspiré du travail de Nicolas Beaufort. Pour valider ces algorithmes une interface de test nommé «algoSim» a été développé.

Lorsque ces algorithmes furent prêts nous avons implémenté un logiciel de pilotage nommé «wbDriver», qui doit permettre à la fois à l'utilisateur de réaliser l'apprentissage mais aussi de pouvoir exploiter cette apprentissage. Pour tester cette interface un logiciel de simulation de l'environnement du wifibot a été crée.

Tester sur WIFIBOT Utiliser la librairie ou écrire une librairie de pilotage du robot. Tester l'apprentissage sur cette plate forme.

6.1.4 Efficacité

Il restait ensuite à démontrer l'efficacité de l'algorithme. Très vite il est apparu qu'il était important de pouvoir visualiser la régression. Nous avons donc utilisé un module d'affichage de courbe 3D. Nos structures de données contiennent quatre dimensions : angle, distance, action et q-valeur. En fixant une action déterminée on est en mesure de tracer la régression. Cet outil devait être doté d'une interface permettant de fixer la valeur des paramètres. Nous nous sommes rendu compte que certains paramètres sont essentiels pour disposer d'une bonne régression.

C'est ainsi qu'est née l'idée d'un outil permettant de réaliser une simulation sur de même données en faisant varier les paramètres et en comparant les résultats obtenus.

6.2 Optimisation

Ce chapitre décrit les améliorations à l'algorithme apportées durant mon stage.

Des états externes Pour ne pas «polluer» l'apprentissage il faut extraire les cas où le robot n'est pas en mesure de détecter la cible. Dans ce contexte on doit également prendre en compte deux cas de figure. Le premier est lorsque le robot effectue un mouvement et «perd» la cible dans ce cas on affecte -1 aux variables de distances et d'angles. Le deuxième cas de figure est lorsque que l'on initie un apprentissage, dans ce cas l'état de départ (angle et distance) est initialisé à -2. On crée autant de points que d'actions et on stocke cette liste d'états externes. Dans le cas où la cible n'est pas vue, il suffira de scruter cette liste et de récupérer l'action qui a la plus grande q-valeur.

Une dimension supplémentaire Lors du calcul de l'enveloppe convexe et de la régression locale pondérée nous avons ajouté une dimension qui a pour valeur 1. Cette dimension permet de centrer l'ellipse sur le «barycentre» des points et non plus sur l'axe des ordonnées.

Détection de la cible Auparavant l'algorithme de recherche de l'image recherche un pixel d'une couleur déterminée puis recherche parmi ses voisins une couleur identique afin d'établir une surface de cette couleur. Cette opération s'effectuait récursivement et pour chaque couleur. Afin d'optimiser la recherche un algorithme de type recherche de séquence est maintenant utilisé.

Le nombre de points d'une interpolation Précédemment lors d'une interpolation on récupérait un nombre de points déterminés. Si la collecte était inférieur à cette valeur le point n'était pas interpolé. Au contraire si on possédait plus de point et que l'on pouvait améliorer la prédiction on utilisait uniquement le nombre de points fixés. Pour éviter cet écueil nous avons ajouter un nombre de points maximum ; ce qui permet à l'algorithme d'optimiser les prédictions.

Changement de la librairie de calcul La librairie de calcul utilisé précédemment, SVDLIB [**?**], présente quelques imprécisions de calculs notamment lorsque l'on souhaite déterminer si un point est dans l'enveloppe convexe ou non. Après avoir repéré cette anomalie, nous avons décidé d'utiliser la Gnu Scientific Library (GSL [**?**]) et obtenu des courbes présentant un meilleur aspect.

Normalisation des entrées Lorsque la nouvelle librairie fut mise en place, il restait un problème l'apparition de «couches» sur la courbe qui semble être lié au fait que les dimensions distances et angles ne sont pas normalisées. La bande passante est unique et s'applique à ces deux dimensions. Pour résoudre ce problème et puisque nous connaissons les valeurs minimum et maximum de angle et distance, nous avons choisi de normaliser ces deux axes.

6.3 Difficultés rencontrées

Lors du développement logiciel j'ai eu quelques problèmes avec l'utilisation de GTK et d'OpenGL. L'utilisation des ces deux technologies est simple avec GTK mais pas avec le wrapper C++. Cette difficulté est certainement née de mon inexpérience dans la programmation d'application en GTK.

J'ai eu des difficultés à comprendre le modèle mathématique utilisé dans smart notamment l'enveloppe convexe. C'est pourquoi j'ai écrit quelques scripts en python pour comprendre ces mécanismes.

D'autres points de complexité sont apparus lors de l'utilisation de l'algorithme de détection de la cible qui utilise un système de récursivité inadapté pour une utilisation avec du C++ (stack overflow). On a donc dû penser à utiliser une autre type de détection basé sur les séquences.

6.4 Comparaison des plannings

Dans cette section nous allons comparer le planning prévisionnel de la section 3.5 et le planning effectif présenté à la page suivante (section 6.5). La première observation est que le développement de la librairie wbLib fut plus longue que prévu. Ce retard s'explique par la volonté de l'équipe d'uniformiser le code pour le rendre compatible et ouvert à d'autres technologies. La seconde concerne un point de vue non identifié en début de projet : comment valider les résultats. Pour cela deux outils ont été créé et on sensiblement modifié les délais prévisionnels. Enfin les tests sur le robot sont long et fastidieux, et malheureusement quelques uns ne sont pas terminés.

6.5 Planning effectif

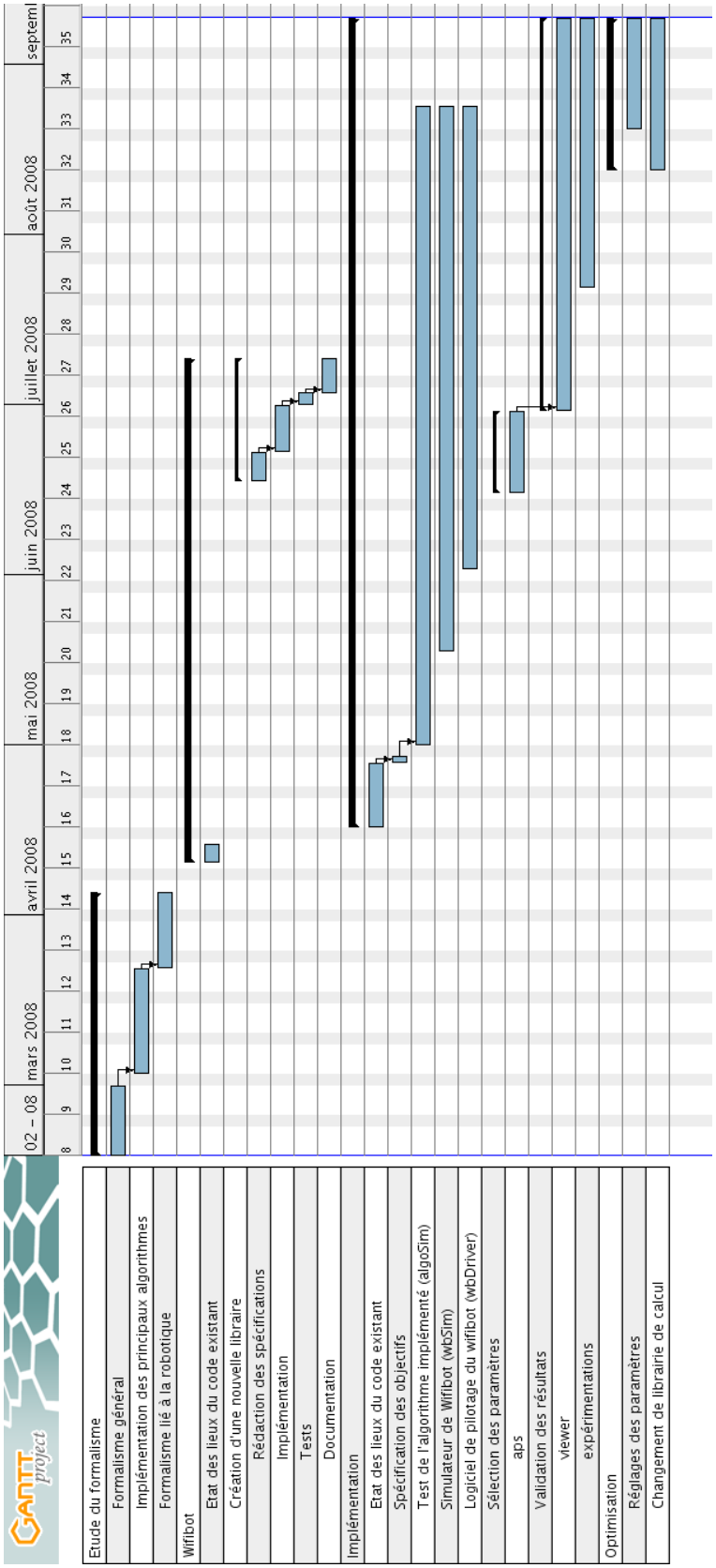


FIG. 6.2 – Planning effectif du projet

Chapitre 7

les livrables

7.1 Cartographie logicielle

La figure 7.1 présente les différents logiciels développés pour le projet. Parmi eux **algoSim** permet de tester les algorithmes à l'aide d'une interface 2D en s'affranchissant du pilotage du robot. **wbDriver** est le logiciel de pilotage du robot qui peut s'interfacer avec la **wbLib** ou **wbSim**. wbSim est une simulation en 3D du wifibot. wbLib est l'API mise en place sur le robot pour pouvoir le piloter. wbDriver et algoSim génèrent un fichier «traces.txt» qui contient toutes les données d'apprentissage. A partir de ces données **aps** permet de générer des graphiques permettant de juger de la qualité des interpolations. Le dernier logiciel **viewer** permet à partir de ce même fichier de représenter la forme de la régression.

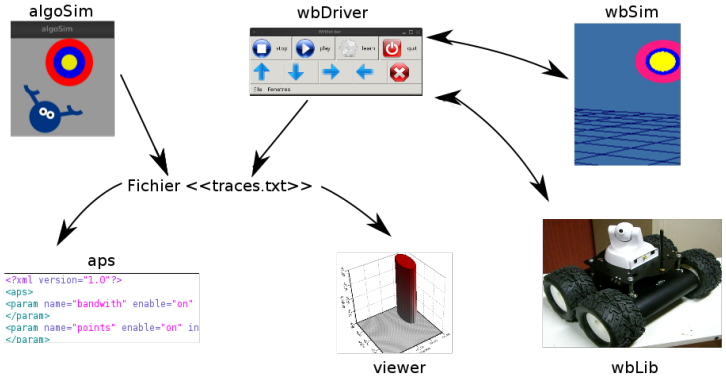


FIG. 7.1 – Cartographie des logiciels

traces.txt A titre informatif, le fichier traces.txt contient les données suivantes :

1. Ancienne distance ;
2. Ancien angle ;
3. Action utilisée ;
4. Nouvelle distance ;
5. Nouvel angle ;
6. Récompense ;

et se présente sous cette forme :

```
0,281250 0,451667 1 0,331250 0,508333 0
0,331250 0,508333 1 0,331250 0,508333 0
0,331250 0,508333 1 0,412500 0,518333 0
0,412500 0,518333 1 0,512500 0,523333 0
0,512500 0,523333 1 0,656250 0,533333 10
```

7.1.1 algoSim

algoSim est un logiciel qui permet de tester les algorithmes sur une interface 2D. Au lancement du logiciel trois fenêtres apparaissent. La première est composée de l'interface 2D : un petit robot et une cible, tel que représenté sur la figure à coté de ce texte. La seconde fenêtre permet d'afficher la courbe 3D des données brutes du kd-tree ou des données interpolées.

La troisième fenêtre présente le nombre de points présents dans le kdtree et des compteurs de type d'interpolation (permettant d'estimer si l'apprentissage est suffisant ou non). Su cette fenêtre apparaît également la prédiction de la prochaine action, c'est à dire la meilleure action de l'état d'arriver. Les données d'entrées de l'algorithme Hedger [**?**] sont réellement la distance et l'angle. Ce logiciel nous a permis de détecter les éventuels bugs d'implémentation des algorithmes et d'appréhender de façon plus significative la régression. L'utilisation d'algoSim consiste à utiliser les touches directionnelles afin de lui apprendre à se diriger vers la cible. Lorsque l'«agent» a suffisamment appris, il doit être en mesure de prédire la meilleure action à réaliser. Un fichier de configuration en XML permet de spécifier la valeur des paramètres d'apprentissage.

7.1.2 wbLib

wbLib est la librairie installé sur le wifibot permettant de le contrôler. Ce contrôle se fait via une connexion TCP/IP. wbLib est composé de trois couches logicielles comme l'évoque la figure 7.2.

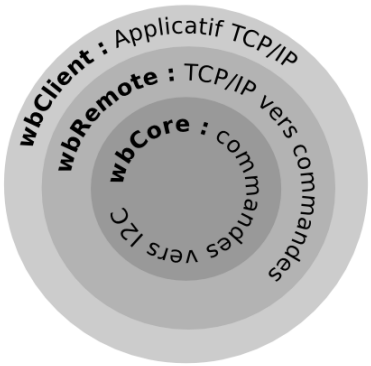
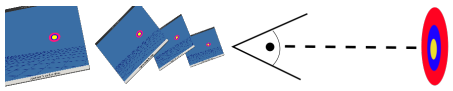


FIG. 7.2 – Architecture logicielle de wbLib

wbCore est la couche la plus près du matériel, elle prend en entrée des commandes ASCII et les transforme en commande I2C. **wbRemote** est la couche réseau, elle permet de faire l'interface entre wbCore et notre applicatif. Les commandes TCP sont plus spécifiques. **wbClient** est la couche la plus abstraite, c'est elle généralement qui contient la logique de l'application. *(La liste des commandes et leur significations peuvent être trouvées sur la documentation de wbLib.)*

7.1.3 wbSim

wbSim est le logiciel qui permet de simuler l'environnement du Wifibot en 3D. Cet environnement se résume à la création d'un objet cible composé de trois cercles concentriques de couleurs déterminées. wbSim contient également deux serveurs TCP/IP. Le permier est un serveur HTTP qui délivre des images JPEG extraites de la scène 3D en flux continu. Le second serveur émule wbRemote puisqu'il permet de contrôler la caméra OpenGL (qui représente la caméra du wifibot).



7.1.4 wbDriver

wbDriver est le logiciel qui permet de réaliser l'apprentissage du robot. Il peut se connecter indépendamment sur le wifibot (via wbLib) ou sur le simulateur (wbSim). Il dispose d'un fichier de configuration «config.xml» composé de deux parties. La première partie concerne des informations de connexion tel que l'adresse IP des serveurs et leurs ports associés. La seconde partie contient la valeur des paramètres des algorithmes (bande passante, nombre de points pour l'interpolation, nombre de points par feuille).

L'interface graphique se compose de deux fenêtres. La première est un rendu de ce que la caméra du wifibot capture. Elle permet également de savoir si le logiciel a détecté la cible ou non. La seconde fenêtre permet de manipuler le wifibot et de changer de mode d'exploitation de l'algorithme. Nous pouvons alors choisir entre exploiter ou explorer. Chaque pas d'apprentissage est constitué des étapes suivantes :

1. Réalisation de l'action (pendant 5s) ;
2. Détection de la cible (pendant 2s) ;
3. Réalisation de l'apprentissage ;

7.1.5 aps

aps signifie *Automatic Parameter Selection*. Comme son nom l'indique, il a été développé afin de fixer les valeurs des paramètres d'un apprentissage sur le Wifibot. Les données d'entrées de ce logiciel sont contenues dans le fichier «traces.txt». Les données de sorties sont multiples : qualité et recouvrement de l'interpolation.

Le principe de ce logiciel est d'interpoler une zone à partir du fichier traces.txt pour une configuration de paramètres données. Pour chacun de ces paramètres on définit une valeur initiale, une valeur finale et une valeur d'incrément. Par exemple avec cette configuration de paramètres :

- nom : bande passante
  - valeur initiale : 0.4
  - valeur finale : 0.5
  - valeur incrémentale : 0.1
- nom : points par feuille
  - valeur initiale : 10.0

- valeur finale : 13.0
- valeur incrémentale : 1.0

Voici les interpolations qui seront jouées :

Index	Bande passante	Points par feuille
1	0.4	10.0
2	0.4	11.0
3	0.4	12.0
4	0.4	13.0
5	0.5	10.0
6	0.5	11.0
7	0.5	12.0
8	0.5	13.0

params.xml Ce fichier définit les interpolations que l'on souhaite jouer. Voici un exemple de fichier de configuration de paramètres d'aps :

```
<?xml version="1.0"?>
<aps>
<param name="bandwith" enable="on" intial_value="0.4" end_value="0.8" step_offset="0.1">
</param>
<param name="points" enable="on" intial_value="10.0" end_value="15.0" step_offset="1.0">
</param>
<param name="pointsmax" enable="on" intial_value="15.0" end_value="15.0" step_offset="1.0">
</param>
<param name="pointsfeuilles" enable="on" intial_value="10.0" end_value="10.0" step_offset="1.0">
</param>
</aps>
```

Pour plus d'information sur ce fichier, veuillez consulter la documentation associée à ce logiciel.

Données de sorties Pour chaque configuration de paramètres, deux images de sorties sont générées. La partie «(A)» de la figure 7.3 présente la meilleure action à réalisée en fonction de la position. L'image doit être représenter de la façon suivante : la distance sur l'axe des abscisses et l'angle sur l'axe des ordonnées. La couleur correspond à l'action «aller tout droit». La couleur blanche signifie «ne rien faire» ce qui peut signifier que l'interpolation est impossible ou que la meilleure action est effectivement de ne rien faire. La partie (B) possède les même caractéristiques de représentation, à ceci près qu'elle représente le statut de l'interpolation. Dans cet exemple Le rouge indique une interpolation normale, le turquoise indique que l'interpolation est en dehors des «bornes» et le vert indique que l'interpolation est hors de la zone IVH (de l'enveloppe convexe).

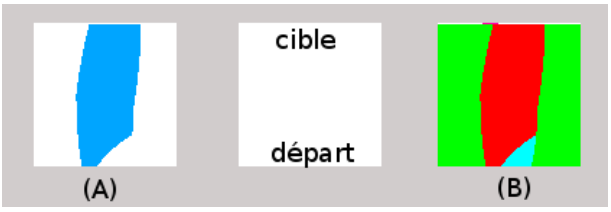


FIG. 7.3 – Un apprentissage simple : «Aller tout droit»

Courbes d'état d'interpolation Le graphique présenté en 7.4 est composé des courbes d'état d'interpolation pour chacune des configurations proposées. On peut ainsi jauger de la qualité d'une interpolation par rapport au fichier d'entraînement fourni. On retrouve sur cette courbe une similarité entre la configuration 0 et les résultats d'interpolation présenté à la figure 7.4.

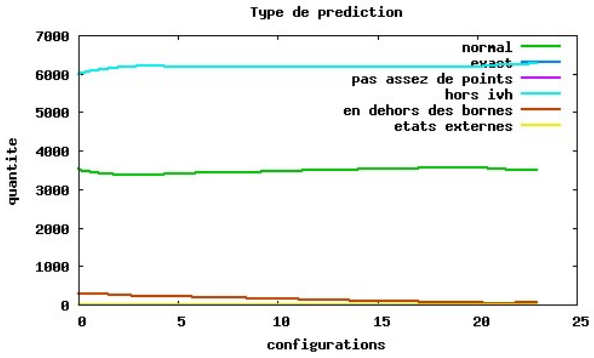


FIG. 7.4 – Etat d'interpolation de chacune des configurations

7.1.6 viewer

viewer est le logiciel permettant de représenter en 3D la régression. Chaque point du kdtree est représenté par 3 axes : angle, distance, action et une q-value. Pour réaliser la représentation en 3D on spécifie une action déterminée.

Ce logiciel utilise la librairie de Nicolas Rougier pour afficher la courbe 3D. Les données en entrées sont contenues dans le fichier traces.txt. Le logiciel permet de sélectionner qu'elles sont les traces à exploiter. Une fenêtre permet également de spécifier la valeur de chacun des paramètres suivants : bande passante, nombre de points par feuille, nombre de points minimum pour l'interpolation, nombre de points maximum pour l'interpolation, α, λ, γ.

La figure 7.5 présente à gauche l'affichage des données brutes du kd-tree. La partie droite affiche la régression que l'on obtient avec l'algorithme Hedger. Le fichier traces.txt utilisé contient 43 points, l'apprentissage réalisé était d'aller tout droit.

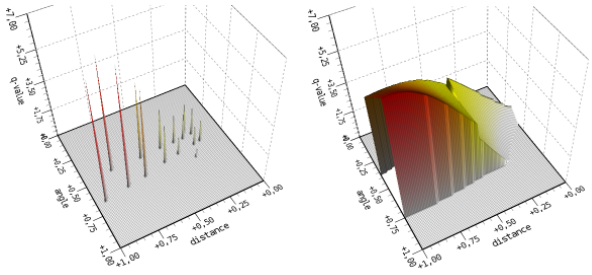


FIG. 7.5 – Impressions écrans de viewer

Chapitre 8

Conclusion

Adopter une approche de gestion de projet dans une thématique de recherche n'est pas une chose aisée. J'ai pu, par le biais de ce stage, appliquer mes connaissances de gestion de projet apprises tout au long de l'année et tenter d'en déduire un modèle adapté à la recherche. Dans notre contexte, la planification du projet est simplifiée parce que le modèle était déjà choisi, à savoir les algorithmes de Hedger [**?**] et Peng [**?**]. Nous pouvons également indiquer qu'il n'y a pas de culture projet dans une équipe de recherche. D'un autre coté il est très difficile de faire une gestion de projet autour d'une hypothèse qui peut se révéler vaine.

Malgré cela le bilan du projet semble positif. L'algorithme est maintenant certifié, des outils d'analyses ont été mis en place, le wifibot dispose d'une librairie prête à être «urbifier» ; la documentation existe. Enfin et c'est le point le plus positif le robot apprend.

J'ai pour ma part beaucoup appris de mon encadrant, Alain Dutech, sur l'apprentissage par renforcement. J'ai disposé de l'expertise de toute l'expérience de l'équipe MAIA pour réaliser ce stage.

Il y a de nombreuses perspectives au projet. Parmi elles il pourrait être intéressant de comparer la régression actuelle avec d'autres types de régression. Nous pourrions également envisager de rendre l'algorithme plus adaptatif à d'autres contextes d'apprentissage ou encore optimiser la structure de stockage. Autant de voies qui seront, je l'espère, approfondies par d'autres stagiaires.

Annexe A

Spécification du Wifibot

Le wifibot est un robot distribué par la société Robosoft. Voici l'architecture matérielle du robot :

- 4 roues codeuses (possédant un système odométrique) ;
- 2 capteurs infrarouge frontal ;
- Une caméra IP possédant deux axes de rotation (PAN/TIL) ;
- deux batteries pour une autonomie d'environ 2 heures ;
- un mesh WIFI ;
- Architecture : Processeur MIPS, 400 Mhz, 64Mb de RAM, 32Mb de flash.

La caméra qui est IP fournit un service HTTP et délivre un flux d'image jpeg continu. Les images peuvent être configuré en taille et en qualité, mais pour des soucis de simplicité le format 640x480 a été retenu. Nous avons également retenu un nombre d'image de 5 par seconde. La caméra présente des temps de latence important et un temps d'adaptation à la luminosité également conséquente. Les roues du wifibot possèdent un système d'asservissement, qui permet d'adapter la vitesse indépendamment et automatiquement pour chaque roue. Le système d'exploitation utilisé est **Linux**, ce qui nous permet d'accéder au shell via ssh et d'envoyer des fichier via scp.

Notons qu'en fonction du niveau de charge de la batterie le comportement du robot ne sera pas identique.

FIG. A.1 – Wifibot

Annexe B

Hedger

Algorithme B.1 : Algorithme d'apprentissage (Hedger Training)

Entrées :
Arbre de points d'entrainements (Kdtree)
Etat de départ $\vec{s_t}$
Action $\vec{a_t}$
Etat suivant $\vec{s}_{t+1}$
Récompense r_{t+1}
Taux d'apprentissage α
Facteur de pondération γ
Nombre de points minimums de la régression k
bande passante de la régression h
Sorties :
Kdtree mis à jour
début
  $q_{min} \longleftarrow \frac{r_{min}}{1-\gamma}$
  $q_{max} \longleftarrow \frac{r_{max}}{1-\gamma}$
  $\vec{x} \longleftarrow (s_t, a_t)$
  $q_{t+1} \longleftarrow$ Meilleure prédiction de Q-value depuis l'état s_{t+1}
  $q_t \longleftarrow$ Prédiction Q-value (s_t, a_t)
  $K \longleftarrow$ Nombre de points utilisés dans la prédiction
  $k \longleftarrow$ Poids utilisés dans la prédiction
  $q_{new} \longleftarrow \alpha(r_{t+1} + \gamma q_{t+1} - q_t) + q_t$
  $S' \longleftarrow S \cup (\vec{x}, q_{new})$
  pour *Chaque point,* $(\vec{x}_i, q_i)$ *dans K* **faire**
    $q_i \longleftarrow \alpha k(q_{new} - q_i) + q_i$
  fin
  retourner S'
fin

Algorithme B.2 : Algorithme de prédiction (Hedger Prediction)

Entrées :
Arbre de points d'entrainements (Kdtree)
Etat s
Action a
Nombre de points minimum de la régression k
bande passante de la régression h
Sorties :
Prédiction Q-value(s, a)
début
  $\vec{x} \longleftarrow (s, a)$
  si $\vec{x} \in extern\_states$ **alors**
    retourner Q-value correspondant
  fin
  $K \longleftarrow$ Points d'entraînements ayant la même action
  si *Pas assez de points d'entraînements (count(K) < k)* **alors**
    retourner $Q_{default}$
  fin
  $H \longleftarrow$ enveloppe convexe (IVH) construit d'après K
  si $\vec{x} \notin H$ **alors**
    retourner $Q_{default}$
  fin
  $q_{s,a} \longleftarrow$ Résultat de la régression locale pondérée
  si $q_{s,a} > q_{max}$ *ou* $q_{s,a} < q_{min}$ **alors**
    retourner $Q_{default}$
  fin
  retourner $q_{s,a}$
fin

Algorithme B.3 : Enveloppe convexe

Entrées :
Liste de points d'entrainements L
Point $\vec{x}_q$
Sorties :
Point $\vec{x}_q$ dans l'enveloppe convexe ou non
début
  Créer une matrice K des points de L
  $V \longleftarrow X(X^T X)^{-1} X^T$
  $H \longleftarrow \vec{x}^T \ (X^T X)^{-1} \ \vec{x}$
  si $H <$ *max de la diagonale de* V **alors**
    retourner vrai
  fin
  retourner Faux
fin

Algorithme B.4 : Algorithme de la régression locale pondérée (LWR Prediction)

Entrées :
Liste de points d'entrainements L
Point dont ont cherche l'interpolation $\vec{x}_q$
Bande passante h
Nombre de points de la régression k
Sorties :
Prédiction de $\vec{x}_q$
début
  $l \longleftarrow$ des k points de L les plus proches de $\vec{x}_q$ triés par distance
  Créer une matrice A contenant les points de l
  Créer une matrice B des Q-values de l
  pour *Tous les points de* l **faire**
    $d_i \longleftarrow$ distance(point de l, $\vec{x}_q$)
    $k_i \longleftarrow$ kernel(d_i, h)
    Multiplier les i lignes de A par les k_i
    Affecter B par la Q-value en cours
  fin
  Résoudre $AX = B$
  si *la pseudo inverse est vide* **alors**
    retourner Moyenne des Q-Values des points d'entrainements
  fin
fin

Annexe C

kd-tree

Algorithme C.1 : Insertion d'un point dans un kd-tree

Entrées :
Point $\vec{x}$ à insérer
Noeud N
Sorties :
début
  si N *est une feuille* **alors**
    Ajouter $\vec{x}$ à N
    si N *est plein* **alors**
      Appel algo «Découpage de feuille»
    fin
  sinon
    si *Si* $\vec{x} <$ *valeur médianne* **alors**
      Appel récursif «Insertion d'un point» dans la branche de gauche
    sinon
      Appel récursif «Insertion d'un point» dans la branche de droite
    fin
  fin
fin

Algorithme C.2 : Découpage d'une feuille d'un kd-tree

Entrées :
Noeud N
Sorties :
début
  $M \longleftarrow$ nouveau noeud avec deux feuilles
  Calculer la valeur médiane
  pour *tous les poins* $\vec{x}$ *de* N **faire**
    Appel «Insertion d'un point» à M
  fin
  $N \longleftarrow M$
fin

Détection de la cible

Algorithme D.1 : Recherche de la cible dans une image HSL 640x480

Entrées :
Image 640x480 HSL
$nbLigne \leftarrow$ nombre de ligne pour valider le motif
Sorties :
La localisation de la cible
début
  $i \leftarrow 0$; $ligne \leftarrow 0$; $LigneOk \leftarrow 0$; $Target \leftarrow 0$
  tant que $i <$ *largeur de l'image* **faire**
    $j \leftarrow 0$
    tant que $j <$ *hauteur de l'image* **faire**
      si *La couleur du pixel image(i,j) est differente* **alors**
        $Seq \leftarrow$ nouvelle séquence
        $Seq.couleur \leftarrow$ couleur image(i,j)
        $Seq.debut \leftarrow i$
        $Seq.fin \leftarrow findecouleurimage(i, j)$
        Ajouter Seq liste des séquence $ListSeq$
      fin
    fin
    pour $Seq \in ListSeq$ **faire**
      $tmpTarget \leftarrow$ RechercheMotif(Seq, «rose, jaune, rose»)
      si $tmpTarget! = 0$ **alors**
        $ligne \leftarrow ligne + 1$
        si $ligne >= nbLigne$ **alors**
          si $tmpTarget > Target$ **alors**
            $LigneOk \leftarrow ligne$
            $Target \leftarrow tmpTarget$
          fin
        fin
      sinon
        $ligne \leftarrow 0$
      fin
    fin
  fin
  retourner localisation($Target$, $LigneOk$)
fin

Script python : calcul de l'IVH

```
import scipy
import scipy.linalg
import pylab
import matplotlib.patches
import math

#### X build from kdtree
X = scipy.array(([0.1, 0.1, 1.0, 1.0],
                 [0.3, 0.2, 1.0, 1.0],
                 [0.15, 0.3, 1.0, 1.0],
                 [0.2, 0.5, 1.0, 1.0]))


XT = X.transpose()
XTX  = scipy.dot(XT, X)
XTXinv = scipy.linalg.pinv2( XTX )
XTXinvX = scipy.dot(X, XTXinv)
V = scipy.dot(XTXinvX, XT)

maxDiag = V[0][0]
for i in range(4):
        if V[i][i] > maxDiag:
                maxDiag = V[i][i]
def calcH(K):
        KT = K.transpose()
        temp = scipy.dot(XTXinv, K)
        H = scipy.dot(KT, temp)
        return H[0][0]

def testPoint(maxDiag, point):
        print calcH(point)
        if calcH(point) <= maxDiag :
                print "le point ", point, "est dans l'IVH"
        else:
                print "le point ", point, "n'est PAS dans l'IVH"

testPoint(maxDiag, scipy.array(([0.25],[0.25],[1.0],[1.0])))
testPoint(maxDiag, scipy.array(([0.5],[0.5],[1.0],[1.0])))
```