Planification Automatique et Techniques d'Intelligence Artificielle

Auteurs :
AULAGNIER Corentin
BOCHATON Mathieu
CHELLE Lea

Supervisé par :
FIORINO Humbert

Master 1 - Informatique
Unversité Grenoble-Alpes, Isere, France

19 avril 2018

Table des matières

1 Introduction

Pour la base de notre code, nous avons repris le code de Lavocat Terrier et Faure. Les packages : controller, motors, sensors, utils et vue étaient déja présents. Nous avons créé deux nouveaux packages : solver et camera. Nous détaillerons plus tard leur fonctionnement.

2 Modification du code de départ

2.1 Prise en main

La quantité de code était importante ce qui le rendait difficile à comprendre au départ. Lorsque nous avons eu une vision globale des fonctionnalités, nous avons modifié certaines parties que nous pensions superflues.

2.2 Class Controller

le package controller a été modifié en grande partie pour prendre en compte pddl4j, et l'exécution des nouvelles actions. Nous avons donc créé la fonction execute() et des ActionsGiver (AG).

2.2.1 execute(Action)

Cette fonction fait exécuter une action par le robot. Son comportement varie en fonction de l'action qui lui est fourni, il y 4 types d'actions :
- Prendre palet : avance jusqu'à toucher un palet, sur un maximum de 0,8s puis ferme ses pinces.
- Lâcher palet : avance sur 0,4s ouvre ses pinces puis recule sur 0,4s (l'idée est de pouvoir ensuite faire une rotation sans être gêné par le palet que l'on dépose).
- Deplacement : x0 y0 x1 y1 : gère TOUS les déplacements.

Déroulement d'un déplacement pour aller chercher un palet :
- Récupère les valeurs x et y de la position actuelle du robot et sa destination (données par l'ActionGiver).
- Calcul la valeur absolue de l'angle à exécuter pour s'orienter vers la destination par rapport à l'orientation actuelle du robot.
- Détermine si la rotation doit s'effectuer vers la droite ou vers la gauche (ce serait une perte de temps de faire plus d'un demi-tour dans un sens).
- Effectue la rotation.
- Commence à avancer.
- Si on croise la ligne correspondant à l'axe X du palet qu'on va chercher, on s'aligne sur cette ligne et on avance sur cet axe (possible uniquement si on va vers un autre X).
- Le déplacement se termine lorsqu'un palet est touché OU au bout d'un certain temps calculé en fonction de la longueur du déplacement et on met à jour la

position actuelle du robot).

Déroulement d'un déplacement pour aller poser un palet dans le camp adverse :
- S'oriente vers le nord (demi-tour vers le camp adverse).
- Avance sur l'axe des X (la ligne de couleur) sur laquelle se trouvait le palet récupéré en ligne droite.
- Le déplacement se termine lorsqu'on atteint la ligne blanche (on met à jour la position actuelle du robot).

L'arrêt sur la ligne blanche lorsqu'on rentre au camp adverse permet de garantir notre position sur l'axe des Y à cet instant. Le réajustement sur la ligne de couleur lorsqu'on va chercher un palet permet de garantir notre position sur l'axe des X à cet instant. La combinaison de ces deux assurances a permis lors de nos tests d'obtenir des résultats satisfaisants en matière de précision.

Amélioration
Il est possible d'ajouter une vérification des lignes de couleurs des axes Y lorsqu'on va chercher un palet : si on arrive à la ligne de couleur correspondant au Y de notre point, on fait un tour sur nous-mêmes pour chercher un palet avec le détecteur. Si on trouve un palet on va le chercher, sinon on rentre au camp. Le plus gros problème que nous rencontrons est lorsque le robot veut récupérer un palet sur le même axe des X que son point de départ. On peut en effet se retrouver dans un cas où le robot est trop décalé sur l'axe des X quand il atteint la ligne blanche et sans l'amélioration proposée ci-dessus on peut boucler à l'infini sur ce palet sans jamais le trouver.

2.2.2 Interface ActionGiver

ActionsGiver est une interface qui permet de récupérer des résultats de pddl. 2 class implémentent cette interface, AGNaif et AGExpert.

AGNaif
Cette class lance pddl à chaque fois que le contrôler en a besoin. Elle récupère les palets présents à ce moment-là sur la table, et cherche les actions pour aller chercher le plus proche.

AGExpert
AGExpert est une class anticipatrice. C'est-à-dire que la première recherche se fait au début de la partie normalement. Mais la recherche suivante est lancée tout de suite après en enlevant de la liste des palets celui qu'on va aller chercher. Lorsque le contrôler à besoin de ses actions il lance la recherche suivante, et récupère les actions déja calculées.
/ ! Cette class est pratique pour accélérer le robot, mais attention car dans le

cas ou le robot n'est pas seul sur la table, il peut choisir un palet déja choisi par l'adversaire.

3 Package solver

Dans ce nouveau package, nous avons créé trois classes : Solver, Parser et ProblemWriter. Le but de ce package est de fournir une interface à la résolution de pddl en lui fournissant une liste de positions de palets et la position du robot.

3.1 Class Solver

Solver est la classe principale. C'est la seule class de ce package visible de l'extérieur. Elle utilise ProblemWriter et Parser pour fournir une liste d'actions valide et la plus simple possible.

3.2 Class ProblemWriter

ProblemWriter ecrit le fichier default.pddl compatible avec notre domaine pddl. Ce fichier change en fonction des palets présents sur le terrain et de la position du robot.

Domaine pddl

Nous avons découpé notre table en :
- 13 lignes [0 ... 12] : 0 correspond à notre droite, et 12 à notre gauche.
- 13 colonnes [0 ... 12] : 0 correspond à la ligne blanche de notre en-but, et 12 à la ligne blanche de l'en-but adverse.

Les actions permettent au robot :
- Se déplacer seulement sur les cases adjacentes à sa position.
- Prendre le palet s'il se trouve à la même position que lui.
- Lâcher un palet s'il en a un et s'il se trouve en Y = 12.

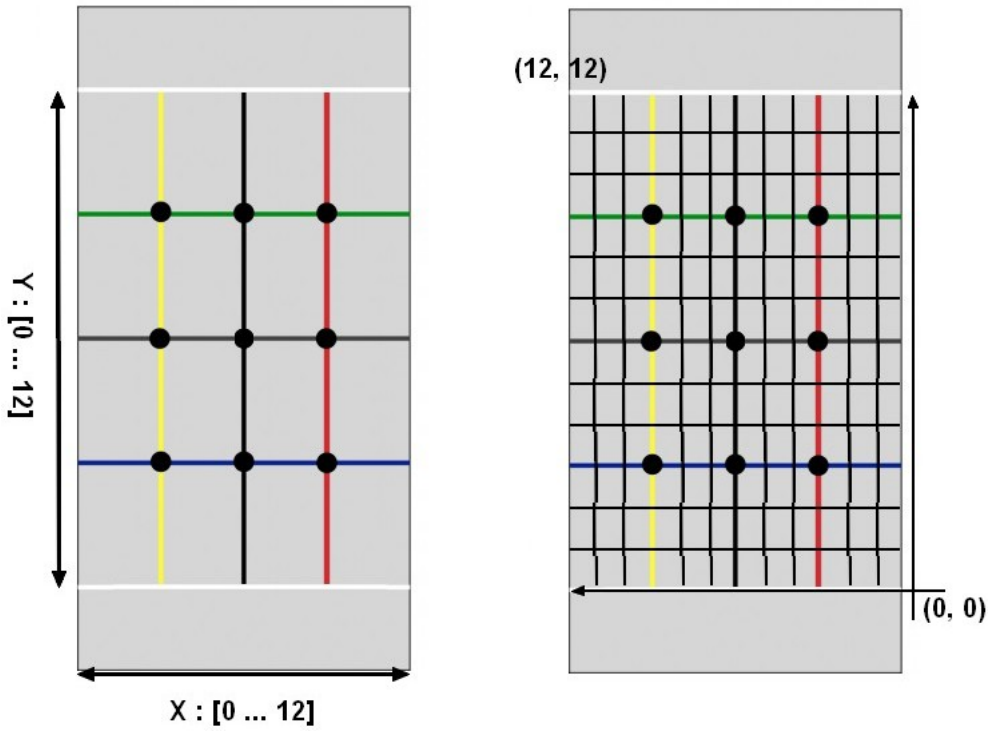

FIGURE 1 – Découpage de la carte

3.3 Class Parser

Parser récupère les actions résultats de pddl, traduit les actions en actions compréhensibles par le controlleur. Puis remplace les déplacements de proche en proche par un seul déplacement d'un point A à un point B.

La liste des actions est donc toujours de la forme :
- 1.Déplacement de A vers B.
- 2.Prendre le palet en B.
- 3.Déplacement de B vers C.
- 4.Lâcher le palet en C.

4 Package camera

Le but de ce package est de gérer la récuperation des positions des palets sur le terrain de jeu.

4.1 Class Server

Cette class a été créée de toutes pieces en s'inspirant du code de Mr Carretero et Mr Chastagner. Ce serveur récupère constamment les positions fournies par la caméra et met à jour cette liste dans la class Caméra. Il y a deux fonctions principales car il y en a une utilisée par la brique pour jouer, et l'autre par notre second main : MainCamera pour nous afficher la position traduite des palets dans le terminal.

4.2 Class Camera

Cette class sert de relais entre le serveur et le controller. Le controller récupère la liste des positions avant de lancer le solver. Le server envoie en continu les messages reçus par la caméra. Lorsque les positions sont reçues, elles sont directement traduites par le Traducteur.

4.3 Class Traducteur

Principe

Il est composé :
- De deux tableaux X et Y tel que : X[i] (resp. Y[i]) indique, pour le iéme axe de notre quadrillage, la valeur moyenne que donne la caméra pour cet axe.
- De deux valeurs fixes deltaX et deltaY avec :

deltaX = Taille du terrain en largeur / ((nombre d'axes - 1) * 2)
= 200 / 24

deltaY = Taille du terrain en largeur / ((nombre d'axes - 1) * 2)
= 300 / 24

Pour traduire les positions (i, j) d'un palets, le traducteur renvoie a et b tel que :

X[a]-deltaX <= i <= X[a]+deltaX
Y[b]-deltaY <= j <= Y[b]+deltaY

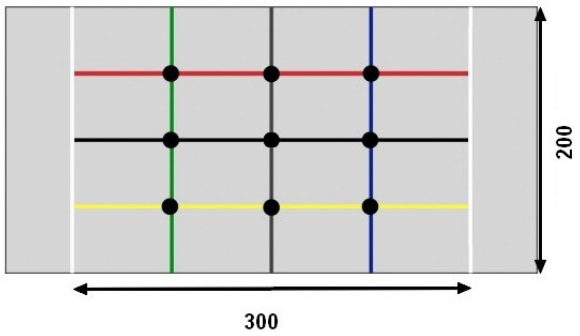

FIGURE 2 – Dimensions de la carte

Comme les points principaux se trouvent à X et Y : 3, 6, 9 il se peut que des tolérances plus grandes soient apportées pour ces valeurs-là.

Calibrage

Pour calibrer notre traducteur, nous positionnons les points comme sur la Figure 3. Grâce à cela, nous avons deux valeurs pour les positions X[0], X[12], Y[0], Y[12]. Nous remplissons donc ces cases avec la moyenne de ces deux valeurs. Puis nous remplissons les autres cases avec la moyenne entre X[i] = moyenne(X[0]+i*Le_décalage, X[12]-(12-i)*Le_décalage) et de même pour Y. Cela nous permet d'avoir une valeur moyenne proche de la réalité. Le point central sert de vérificateur, et affiche un warning si les points ne sont pas cohérents.

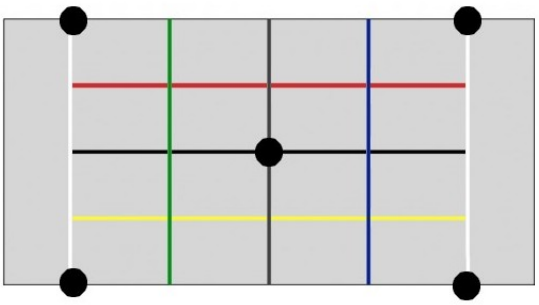

FIGURE 3 – Position des points pour le calibrage de la caméra

5 Réflexions personnelles

5.1 Les versions de Java & PDDL

Nous avons eu beaucoup de mal à prendre en main la brique, et à exécuter pddl sur la brique. Nous avons essayé toutes les versions de pddl, avons modifié plusieurs fois cette librairie, avant d'arriver à quelque chose qui s'exécute et donne des résultats cohérents.

5.2 Temps de réflexions de la brique

Pddl met quelques secondes à s'exécuter avec AGNaif car nous le lançons à chaque fois que nous ramenons un palet dans l'en-but adverse. Cela nous fait perdre beaucoup de temps. Mais il est risqué d'utiliser AGExpert, surtout pour la compétition. Nous avons essayé d'optimiser le code au maximum.

5.3 Organisation du groupe

Nous sommes un groupe de 3 personnes. Nous nous sommes séparés en 2 groupes :

- Léa et Corentin se sont occupés des parties solver et camera.
- Mathieu s'est occupé de prendre en main la brique, le plugin eclipse et l'exécution des actions par le robot.