UNIVERSITÉ Grenoble Alpes

Rapport final
Initiation à l'intelligence artificielle

Leander Feppon, Bilal Lahmami, L3 MIASHS 10/12/2018

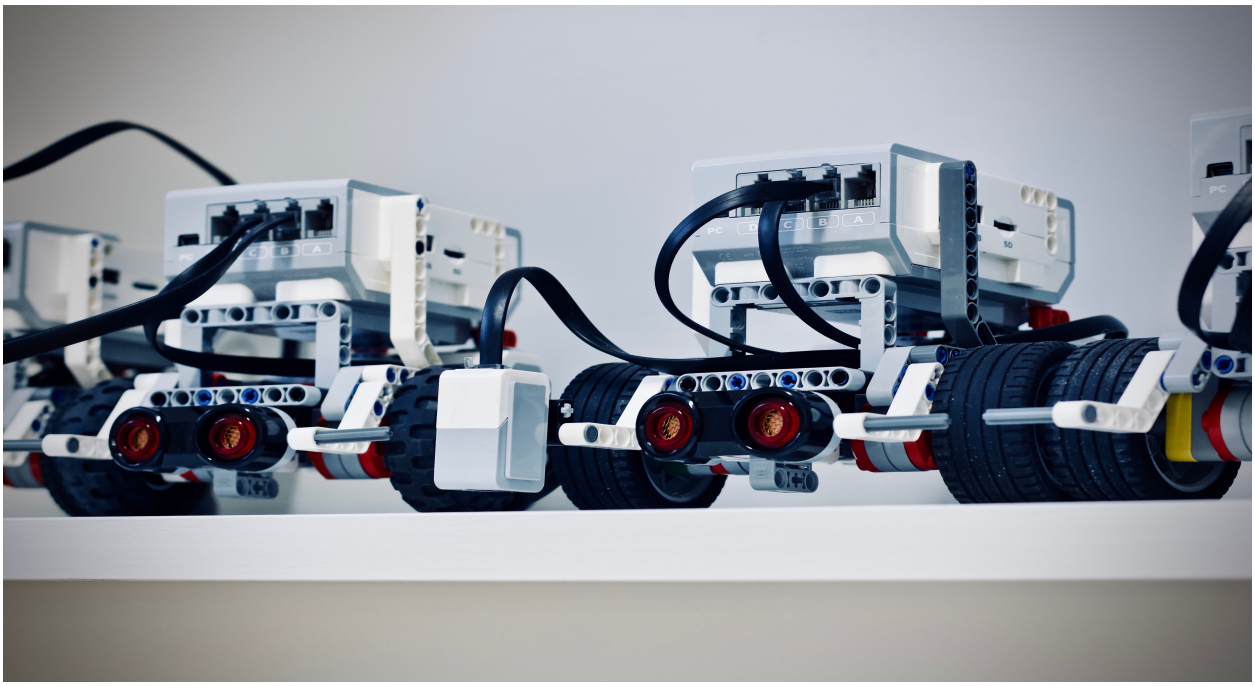

Table des matières

1 Introduction

Le projet arrive à sa fin. Il est l'heure de procéder à notre introspection, et d'en retirer les enseignements. Nous détaillons ici la démarche et les étapes du développement de notre robot. Ces dernières ne sont pas toujours en accord avec notre plan de développement.

Nous essayons de comprendre les raisons de ces écarts, et quelles mesures nous pourrons prendre à l'avenir pour essayer de les minimiser.

2 Guide de lecture

2.1 Pour le client

Ce rapport lui permet de vérifier que ses besoins ont bien été remplis, dans les délais impartis. Si ce n'est pas le cas, ce rapport se devra de lui en expliquer les raisons.

2.2 Pour le binôme

Revenir sur le développement du robot nous permet de voir où et quand nous avons perdu de vu notre objectif, ainsi que d'examiner avec un peu de recul le pragramatisme des solutions apportées.

3 Premiers essais

3.1 Premier projet

La première chose à faire a été de télécharger LeJOS, ainsi que JDK7. IntellIJ dispose d'un plugin LeJOS. Il est nécessaire de préciser le chemin vers LeJOS_HOME dans les paramètres.

Le remote debugging n'est cependant pas disponible, contrairement à Eclipse.

3.2 Familiarisation avec la documentation

Plusieurs ouvrages existent pour se familiariser avec la documentation de cet API. Notre préféré est *Beginning Robotics Programming in Java with LEGO Mindstorms* de Wei Lu, ainsi que la documentation officielle.

3.3 Manipulation des capteurs

Nos premiers essais portent sur les capteurs du robot. Tout d'abord comprendre comment les initialiser puis renvoyer les valeurs sur l'écran de la brique pour tester leur fonctionnement. Nous avons commencer par tester le capteurs d'ultrasons et les distances qu'il pouvait renvoyer et nous sommes vite rendus compte de la fiabilité quelque peu incertaine de celui-ci ce qui nous as rendu sceptique quant à l'utilisation précise que nous comptions en faire. En effet les distances n'étaient pas toujours renvoyées, au delà d'une certaine distance le capteur ne renvoie plus de valeur. En dessous de la distance maximale renvoyée par le capteur d'ultrasons, les valeurs étaient quelques fois changeantes et incertaines également.

Nous avons par la suite testé le capteur de couleurs sur les différentes couleurs du terrain en mode ColorID pour obtenir les valeurs correspondantes de chacune des lignes du terrain ainsi que du gris du sol du terrain. Malheureusement le capteur en mode ColorID ne renvoyait pas non plus des valeurs de manière cohérente, certaines couleurs renvoyaient les mêmes valeurs et certaines valeurs changeaient d'un test au suivant. Se servir du mode ColorID n'était donc pas une option. Pour cette raison, nous avons essayé d'écrire une méthode de calibration du capteur de couleur à exécuter en début de séance. Nous avons donc pris un certain nombre de valeurs RGB pour chaque couleur, que nous avons toutes mises dans un tableau. Nous avons par la suite fait des moyennes sur ces valeurs pour finalement établir des intervalles pour les valeurs de RGB dans lesquels une

couleur captée devaient rentrer pour être reconnue comme telle. Cependant, certains intervalles se chevauchaient, même en prenant des intervalles relativement petits, et donc les couleurs captées n'étaient pas toujours justes.

Finalement le test du capteur de pression donnait des résultats suffisamment consistants pour en faire une utilisation normale dans le contexte de ce projet. Il est à noter également que nous avons remarqué par la suite que tous les aléas et inconsistances au niveau des capteurs et moteurs s'améliorent lorsque la batterie du robot est chargée à 100% et s'aggravent rapidement à mesure que la batterie se vide.

3.4 Manipulation des moteurs

Après avoir compris le fonctionnement des capteurs nous avons expérimenté quelques commandes liées aux moteurs, en faisant avancer le robot d'une certaine distance, ouvrir et fermer les pinces. Nous nous sommes familiarisés aux méthodes permettant de tourner et avons évalué combien de degrés devaient êtres entrés en paramètres pour faire tourner le robot d'un certains nombre de degrés mesurés.

Ici encore un écart entre la valeur entrée en paramètre et celle de rotation effectuée en réalité se fait constater. Sur les lignes droites, nous remarquons aussi que les roues ne sont pas tout à fait synchronisées et que le robot tourne donc légèrement sur la ligne droite malgré l'utilisation d'un MovePilot qui nous permet de synchroniser les moteurs des roues. Ce problème va se montrer plus ou moins marqué en fonction des séances au fil du projet. Nous avons également changé de carrosserie de robot à la seconde séance à cause de ces différents problèmes, sans constater d'amélioration significative.

4 Approfondissement du sujet

4.1 Stratégie

Notre première stratégie principale pour attraper les palets consistait à, lorsqu'en position de recherche d'un nouveau palet, le robot va tourner sur lui même avec le capteur d'ultrasons en activité pour estimer quel palet est le plus proche de celui-ci. Une fois le palet optimal trouvé, le robot se replace en face de celui-ci pour aller le chercher et le ramener, puis il réitère ce comportement en chercher un autre. Cette stratégie n'est pas allée jusqu'au bout est n'est donc pas entièrement détaillée car beaucoup de problèmes n'ont pas étés traités, problèmes qui entre autre nous ont justement fait changer de stratégie par la suite.

Nous nous reposons à présent sur les coordonnées des palets. En utilisant un PoseProvider pour retenir la position du robot, et en pré-enregistrant les coordonées des palets dans notre programme, nous sommes plus efficaces dans la recherche de notre palais. Cependant, si jamais le capteur de touche ou d'ultrasons détectent un palais, le robot est capable de mettre à jour sa position.

Le premier palet ayant une importance particulière, nous ne perdons pas de temps et pré-programmons sa récupération et sa dépose.

4.2 Tachomètre

Comme explicité précédemment, les méthodes de rotation ne sont pas assez précises pour notre usage. En passant par le tachomètre, nous pouvons établir une correspondance entre le sa mesure et le degré de rotation.

Pour cette raison nous avons commencé à réaliser des tests sur plusieurs tours de robots pour plus de précision, en utilisant la valeur du tachomètre pour revenir à la position de départ. Cela pouvait également servir de boussole pour savoir vers quel côté est tourné le robot. Seulement ici aussi il ne suffisait pas de simplement de faire une rotation inverse de la valeur indiquée par le tachomètre, il a donc fallu pour trois vitesses de rotation différentes établir une petite relation entre le degré de rotation entré en paramètre et combien de la valeur du tachomètre il fallait tourner en retour.

4.3 Pince

Suite à cela nous avons essayé de traiter un problème que nous avions avec les pinces. Nous avons cherché à pouvoir ouvrir et fermer les pinces. Si ces dernières étaient déjà dans un état identique à la commande entrée, la commande devait être inefficace.

Il n'était pas évident de trouver les bonnes valeurs d'ouverture/fermeture des pinces et nous avons donc essayé d'utiliser un threshold sur celles-ci pour bloquer l'ouverture ou la fermeture lorsque celle-ci commence à forcer sur les moteurs. Nous n'avons pas réussi à faire fonctionner cela correctement. Cette démarche devait être utile lorsque le programme quittait de manière non prévue et laissant ainsi les pinces ouvertes. Nous nous sommes rabattus sur un simple booléen retenant l'état de la pince.

4.4 Threads

Nous avons cherché ici à résoudre un problème d'exécution parallèle : ouvrir les pinces ne doit pas empêcher le robot d'avancer et inversement. Bien que les threads ne soient pas réellement du parallélisme, et que processeur ARMv5 soit limité à 1 thread, cela suffit pour notre usage.

4.5 RangeFeatureDetector

Ici, nous disposons d'une manière plus efficace de prendre des mesures. Il suffit de préciser un RangeFinderAdapter, la mesure maximale et le durée entre chaque mesure, pour avoir un capteur qui fonctionne.

4.6 Navigator et Pathfinder

La création d'un Pilot entraîne automatiquement la création d'un objet Navigator et d'un PoseProvider, responsable de retenir la position du robot sur le terrain. Cependant, cela n'est efficace que pour des mouvements simples et complètement réalisés.

Le Pathfinder nous permet d'aller à un point A à un point B sans rentrer en collision avec les obstacles donnés en paramètres, ici les palets.

Notre seconde stratégie consiste donc à utiliser une map plutôt que de rechercher les palets sur le terrain. La map nécessite d'entrer des paramètre de taille et s'utilise comme un repère orthonormé

sur lequel on place le robot au point (0,0). Après avoir initialisé celle-ci, on peut y ajouter la position de chacun des neufs palets, entourés chacun d'un périmètre de sécurité. Ce périmètre permet au robot d'éviter les palets qu'il ne veut pas attraper lorsqu'il se déplace sur la carte. La map est dotée de base d'un algorithme de pathfinding qui permet au robot de trouver le chemin le plus rapide pour aller d'un point A à un point B. En plus de cela nous avons ajouté des méthodes qui nous sont utiles dans le contexte du projet.

Cette approche a été décevante car le PoseProvider n'est pas capable de suivre correctement les mouvements du Navigator. Le capteur de couleur n'est pas assez précis pour nous permettre de nous repérer sur le terrain et donc de corriger la position du robot.

Une solution serait d'écrire notre propre PoseProvider et MoveListener et de le passer en paramètre à la création du Navigator.

Nous avons également essayé d'utiliser le FourWayGridMesh ainsi que AstarSearchAlgorithm en conjonction. Les résultats sont encore plus aberrants.

4.7 Arbitrator et Behaviors

Nous sommes partis au début sur une simple variable retenant l'état du robot, et à l'aide d'un switch et d'un évenement, changer d'état en fonction. Nous avons par la suite découvert les classes Arbitrator et Behavior.

Une classe qui implémente la classe Behavior définit les méthodes takeControl(), action() et suppress(). takeControl() indique à l'Arbitrator quand est-ce que ce comportement doit prendre le contrôle. action() est la méthode à effectuer. suppress() la méthode permettant de stopper l'exécution du comportement si un autre comportement avec une priorité plus importante a été trouvée. L'Arbitrator prend un tableau de Behaviors en paramètre, et crée un thread pour chaque élément. Une priorité est ensuite associée à chacun de ces threads.

Si aucun Behavior ne peut prendre le contrôle, l'arbitrator s'arrête à moins que le deuxième paramètre de son constructeur ait été initialisée à vrai. Mais dans ce cas, le programme peut juste freeze.

Manier plusieurs threads n'est pas chose aisée.

5 Réécriture

5.1 IntelliJ et remote debugging

Notre IDE n'a introduit le remote debugging qu'à partir de décembre 2018. Il nous était donc nécessaire de passer par la GUI des outils fournis par LeJOS pour envoyer le programme sur la Brick et l'exécuter. La récupération des erreurs se fait via le fichier d'extension .err. L'outil est cependant buggé, et la fenêtre de récupération/sélection de dossier/fichier s'affiche régulièrement de manière intempestive, ou freeze sans raison apparente.

Nous avons donc décidé d'écrire notre propre programme, pour envoyer, exécuter, et récupérer l'output de notre projet. Ce programme est intitulé LeJosFileSender. Il est fourni en supplément avec le code source. Nous tirons parti de la classe RemoteMenu pour chercher un robot sur le réseau et lui envoyer et lui faire exécuter un programme. Cet outil marche assez bien si lancé depuis Windows. Linux ne semble pas pouvoir détecter les robots. Un autre problème réside dans la recherche du robot directement à l'aide de son IP. Mais notre programme marche suffisament pour notre usage.

5.2 EV3Dev

À la moitié de ce projet, nous avons décider d'utiliser EV3Dev. LeJOS n'est plus maintenu depuis 2015 et nous force à utiliser Java 7.

Nous utilisons particulièrement ici EV3Dev-lang-java qui se base sur l'API LeJOS et permet de transférer notre programme d'une API à l'autre sans trop de problème.

EV3Dev rencontre cependant un problème de temps de chargement. Un simple "java -version" peut prendre une dizaine de secondes et un programme plusieurs minutes. Cela est frustrant.

5.3 EV3Dev-C

Agacés par cette perte de temps et la lourdeur de la JVM, nous avons décider de réécrire le programme en C, ce qui a le plus de sens sur un système embarqué. Et effectivement, notre programme peut s'exécuter instantanément. Bien que l'environnement soit peut-être un peu plus compliqué à mettre en place.

À deux semaines de la fin du projet, devoir tout réécrire a été un challenge. EV3Dev-C étant relativement bas-niveau, il a fallu écrire par exemple notre propre méthode de rotation.

Notre programme est cependant rapide à charger en mémoire, et nous gagnons en mémoire et vitesse de calcul. Le capteur de couleur est aussi mieux calibré sur la dernière version de EV3Dev, et nous pouvons donc utiliser notre stratégie.

Le robot voit toujours sa position calculée à tout moment. Nous utilisons l'algorithme A* implémentée par libastar pour calculer un chemin sans obstacle.

Les résultats quelques jours avant la compétitions semblent être satisfaisants lorsque le robot se déplace assez lentement. Les roues ne sont pas encore totalement synchronisée, et le robot dévie de sa trajectoire en allant trop vite.

6 Bilan

Malgré beaucoup de problèmes, autant hardware que software, nous avons trouvé un moyen de faire exécuter au robot sa tâche. Au final, les aléas liés au matériel ne peuvent pas être contrôlés totalement et certains problèmes reviennent toujours mais il est possible à terme de faire jouer le robot sur la carte à vitesse raisonnable.

De manière générale, les groupes étaient plus ou moins dans un esprit d'entraide, ce qui a permis d'échanger des connaissances et solutions à certains problèmes, mais globalement, tous les groupes ainsi que le nôtre, ont tout de suite commencé le travail sans réellement savoir vers où cela allait les mener et peut être qu'une réflexion plus poussée sur chaque décision ou action aurait permis un gain de temps.

Au final, nous voyons que beaucoup de documents ont été écrits mais que ces derniers n'ont pas forcément été respectés. Travaillant sur une librairie inconnue, du matériel inconnu, et sur notre premier réel projet, nous avons eu du mal à poser des standards et des objectifs concrets, ainsi qu'à les respecter.
Si les choses étaient à refaire, nous prendrions vraiment le temps de réfléchir à l'architecture du projet, une architecture de "haut en bas", plutôt qu'un empilement de "bas en haut" comme nous venons de le faire ici, sans même le réaliser.

Il serait raisonnable de penser à utiliser un gestionnaire de versions pour nos futurs projets.

Ce projet a quand même été une expérience enrichissante, nous poussant à en apprendre plus sur les technologies utilisées et les limites de nos connaissances, mais aussi sur l'importance d'une bonne documentation et d'une bonne réflexion préalable.

7 Table des illustrations

Liste des illustrations

8 Références

[1] EV3Dev, http://www.ev3dev.org/

[2] EV3Dev-C, http://in4lio.github.io/ev3dev-c/

[3] libastar, https://www.bedroomlan.org/projects/libastar/

[4] Damien Pellier, Guide du projet, http://lig-membres.imag.fr/PPerso/membres/pellier/teaching:ia:project

[5] Leander Feppon, Bilal Lahmmai, *Cahier des charges*, 2018

[6] Leander Feppon, Bilal Lahmmai, *Plan de développement*, 2018

[7] Leander Feppon, Bilal Lahmami, *Code source et documentation interne*, 2018