Documentation du code du robot Sunstorm, Persycup 2015

L'équipe Sunstorm
R. Jakse, F. Mollin, R. Boutonnet, Xu Chi, Huanan Sun

22 mai 2015

1 Introduction

Dans ce document, nous introduisons le code qui a servi pour la persycup 2015 sur le robot de notre équipe, SunStorm. Au départ sophistiquée, avec un fil d'exécution pour une file d'événement, un autre pour gérer les moteurs et un autre pour gérer un automate, avec une gestion des coordonnées du terrain et plusieurs mécanismes d'abstractions pour rendre l'écriture de l'automate simple, il s'est avéré que cette architecture était compliquée et en demandait trop au robot qui finissait par avoir un comportement erratique et lent. Le code a été très simplifié et le comportement du robot a été régit par un graphe d'appel de fonction extrêmement facile à suivre, avec surveillance des capteurs et gestion des moteurs aux endroits nécessaire. La gestion directe des capteurs au lieu d'être événementielle a rendu certaines parties du code un peu plus lourde, notamment le suivi de ligne, mais a rendu l'ensemble beaucoup plus simple.

Ce code nous a permis d'arriver en demi-finale lors de la coupe Persycup 2015, et aurait peut-être pu nous faire aller plus loin si tout s'était bien passé lors du départ du robot pour les dernières manches. Malheureusement, la gestion du départ manque de robustesse car celle-ci a été adaptée le jour même de la compétition, pour nous conformer à une règle que nous avions oublié de prendre en compte (départ devant une ligne).

Ce document s'articule en deux parties : le mode d'emploi et l'explication de l'architecture du code et de la stratégie plus en détails.

2 Utilisation

Pour utiliser ce code, vous aurez besoin de mettre en place leJOS sur votre robot Mindstorm. Ensuite, il vous faudra mettre en place votre environnement de développement (Eclipse) avec l'extension leJOS, qui permet de contrôler le robot avec le langage Java et avec des interfaces graphiques et d'envoyer des programmes au robot. Ensuite, vous devrez importer le projet dans Eclipse.

- Installer le robot :

http://sourceforge.net/p/lejos/wiki/Installing leJOS/

- Préparer l'environnement de développement :

http://sourceforge.net/p/lejos/wiki/Installing the Eclipse plugin/

Le projet comporte deux programmes : `Main`, écrit avant la compétition, et `Main2`, adapté le jour de la compétition pour respecter la règle qui impose de commencer devant une ligne.

Tous les paramètres des programmes sont dans une classe de configuration spéciale, ne comprenant que des constantes. Pour configurer le programme `Main`, modifiez le fichier `Config.java`. Pour configurer `Main2`, il faut modifier le fichier `Config2.java`.

La première chose à configurer est le câblage. Vous devez vous assurer d'associer les bons ports au bon capteurs et aux bons moteurs. Le reste des paramètres nécessite une bonne connaissance du code. Ces paramètres permettent de régler finement le comportement du robot.

Pour lancer le programme, vous devez, après avoir configuré les ports dans `Config2.java`, lancer le programme `Main2` sur le robot. Pour lancer le programme, n'oubliez pas :

- d'allumer le robot
- de le connecter par USB ou de l'appairer en Bluetooth avec votre ordinateur
- d'activer la connexion filaire créée par le robot (peut-être fait automatiquement si branché par USB selon votre système d'exploitation)
- de vous assurer qu'aucun programme ne tourne sur le robot (dans le doute, vous pouvez cliquer sur le bouton « Stop Program » de l'interface `EV3Control` trouvée dans le menu « leJOS EV3 → Start EV3Control » d'Eclipse).
- de vous assurer que la calibration est désactivée si vous n'avez pas besoin d'une nouvelle calibration, ou qu'elle est activée si vous en avez besoin.

Vous aurez peut-être besoin de calibrer les couleurs du robot. Pour cela, positionnez la variable de configuration `CALIBRATE_COLORS` à `true` dans Config2. La calibration n'est pas gardée, elle n'est utilisée que pour cette exécution du programme. Si vous souhaitez conserver votre calibration, copiez-collez la sortie de la console correspondant à la calibration (dans l'interface `EV3Control`) dans la variable `colorsCalibration`.

Lorsque vous lancez le programme sur le robot, patientez un petit moment, et vous pourrez régler l'ouverture des pinces (elle doivent être ouvertes à 180°), calibrer les couleurs (si vous avez besoin de le faire) et de choisir de quel côté vous allez placer le robot (sur la ligne de gauche ou de droite). Attention, le robot avance pendant deux secondes avant de choisir la ligne devant laquelle le robot se trouve. Cela permet de débloquer les moteurs et d'éviter que le robot ne parte de travers, ce qu'il faisait de temps en temps sans ce mécanisme.

Le programme ne s'arrête jamais. Vous devez utiliser l'interface EV3Control pour stopper le programme. Si le terrain faisait dix lignes, le robot parcourrait les 10 lignes, sauf problème.

3 Fonctionnement du code et stratégie

3.1 Début de la partie

Au moment où le programme se lance, la méthode statique main est appelée. Elle se charge :

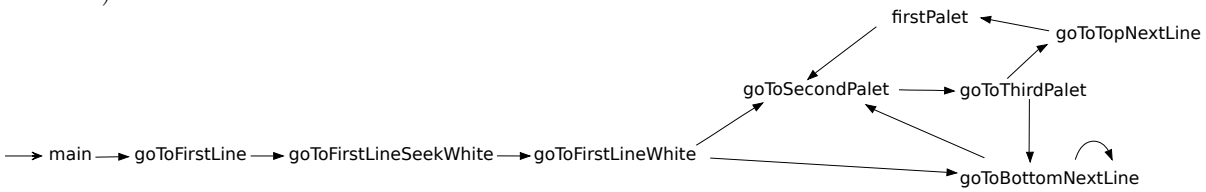
- d'initialiser les moteurs et les capteurs (méthode `initMotorsSensors`). Nous avons écrit une classe pour chaque capteur qui se charge de le gérer de notre manière (sonar : `Sonar`, capteur de couleurs : `ColorSensor`, capteur de palet `TouchSensor`). Pendant cette phase, on pourra calibrer les couleurs.
- d'avancer pendant deux secondes (pour débloquer les moteurs)
- de choisir de quel côté du terrain on se place (droite ou gauche)
- de lancer la partie.

3.2 Stratégie globale

Au départ basée sur des coordonnées absolues et une classe qui choisissait elle même en fonction des coordonnées de départ et d'arrivée s'il fallait faire un suivi de ligne, et sur une classe gérant automatiquement un automate qui recevait des événements basés sur les capteurs, nous avons abandonné l'idée qui était trop complexe à la fois pour le robot et pour nous, pour nous concentrer sur des mécanismes plus simples avec moins d'abstractions.

La partie est organisée en plusieurs petites étapes, chacune représentée par une courte fonction. On pourrait voir cela comme une sorte d'automate écrit de manière fonctionnelle, passant d'un état au suivant quand toutes les étapes de l'état actuel ont été effectuées. Ces actions comprennent la surveillance des capteurs, qui se fait plus ou moins selon un schéma d'attente active, mais pas tout à fait : on continue quand même à effectuer des étapes. Par exemple, on continue de suivre la ligne que l'on est en train de suivre, mais en même temps on vérifie qu'il n'y a pas un palet.

On peut construire un graphe à partir des appels de fonctions (la flèche « A → B » note la relation « A peut appeler B ») :



Notre stratégie est la suivante :

- On prend le premier palet qui est en face de nous, on fonce vers le camp adverse en se décalant pour éviter de heurter les palets qui sont sur la même ligne. Après avoir marqué le premier palet, on se place sur le haut de la ligne de départ (coté camp adverse), et on parcourt la ligne en faisant un suivi de ligne avec le capteur de couleur. Lorsque l'on rencontre un palet, on le récupère et on l'envoie au but, et on recommence sur la même ligne. Lorsqu'on a pris tous les palets de la première ligne :
  - soit on en a pris 3, et on se place sur le haut de la ligne suivante, en continuant de la même manière.
  - soit on a atteint la ligne blanche, et on commence du bas de la ligne suivante. Dans ce cas, lorsque l'on attrape un palet, on revient sur la ligne précédente et on envoie le palet dans le camps adverse, et on revient en haut de la ligne actuelle.

Il n'y a pas de cas d'arrêt, de toute façon on arrête le robot manuellement lorsque la partie est terminée.

3.3 Gestion des ennemis et récupération d'erreurs

Comme toutes les équipes présentes à la Persycup 2015, nous ne gérons pas l'ennemi. En revanche, en raison d'un robot défectueux qui causait des coupures intempestive des capteurs pendant plusieurs secondes de façon aléatoire, nous avons mis en place plusieurs mécanismes de récupération d'erreurs, en particulier :

- Lorsque l'on doit franchir une ligne blanche, si on faisait un suivi de ligne, si on est dans le gris depuis trop longtemps, on « devine » que l'on a franchi la ligne blanche. Plus précisément, pendant le suivi d'une ligne, si on se retrouve dans le gris, on ouvre un angle de plus en plus grand pour retrouver une ligne perdue. Quand cet angle devient grand, on avance un peu pour ne pas risquer de tomber sur une éventuelle ligne blanche que l'on aurait franchie mais manquée pour ne pas risquer de la suivre (lors du suivi de ligne, on suit n'importe quelle couleur qui n'est pas du gris), et quand l'angle devient assez grand, si on n'a pas retrouvé de ligne, c'est qu'on est dans un but. On dépose le palet qu'on a dans les pinces si c'est le cas, on recule et on repart. Le sonar peut également nous informer que l'on est proche d'un mur. Dans ce cas, on recule et on retente de franchir la ligne blanche. Un effet secondaire est que si l'on suit un ennemi de trop près, on recule et on réavance. Cela a pu être observé lors de la compétition.
- Lorsque l'on doit changer de ligne et qu'on traverse le terrain dans une zone grise, si cela fait trop longtemps, on devine que l'on a manqué la ligne. On revient alors en arrière et on retente de traverser la ligne.

4 Conclusion

Ce projet nous a appris qu'un code bien structuré mais complexe et contenant trop d'abstractions peut rendre les choses compliquées à suivre, mal fonctionner, et trop difficiles à gérer (comprendre, écrire, modifier, améliorer). Alors qu'un code simple peut rendre les choses claires et efficaces, quitte à ajouter de la complexité et des abstractions

quand elles sont vraiment nécessaires. Le fait que l'on n'utilise qu'un seul fil d'exécution rend les choses plus faciles à comprendre : tout est séquentiel, et il n'y a pas de risque de problèmes de concurrences. Avec un peu de travail sur le départ, ce code est probablement tout à fait valable et suffisamment robuste pour participer à une nouvelle compétition similaire et avoir des chances d'être bien classé. Avec un robot qui n'a pas de défaut de capteurs, il est certainement possible d'adapter la vitesse pour avancer plus vite. Mais si cela était à refaire, nous partirions certainement avec une stratégie basée sur le sonar.