Projet Lego EV3

Lefranc Joaquim, Skoda Jérôme

Introduction : problématique

Énoncé du problème :

Le robot doit pouvoir suivre une ligne courbe, fracturée, et de couleur arbitraire non constante. Il doit également être capable de s'arrêter automatiquement après avoir accompli deux tours. Le début du circuit est symbolisé par une couleur différente.

Intérêt du problème :

La généralisation de ce problème étant la capacité d'un robot à réagir de lui même face à son environnement grâce à des capteurs. La reconnaissance, le traitement des données et enfin le choix de la décision. De façon plus particulière, le suivi de ligne peut-être utilisé dans des applications de transport : déplacement des bagages dans les gares, organisation automatique d'un parc de Bus «suiveurs», grâce à des parcours tracé au sol.

Fonctionnalités

Utilisation standard :

L'application peut-être lancée de trois manières différentes. Cette sélection se fait grâce aux paramètres : ***-c -s****. Le paramètre* ***-c*** *correspond au mode de calibration permettant d'enregistrer des couleurs dans le fichier* ***calibration.calib****. Ensuite le mode scan disponible avec* ***-s*** *permet de scanner des couleurs pour tester. Enfin sans arguments l'application se lance en mode normal et le robot exécute la fonction de suivis de ligne.*

Fonctionnalités du robot :

Pour commencer le suivis, il faut d'abord placer le robot de préférence à droite de la ligne à suivre et le plus proche de celle-ci, tout en laissant le capteur sur le fond. Le robot va d'abord analyser la couleur du fond et l'enregistrer. Ensuite il pivote vers la gauche pour trouver la ligne et la suit.

Architecture et conception

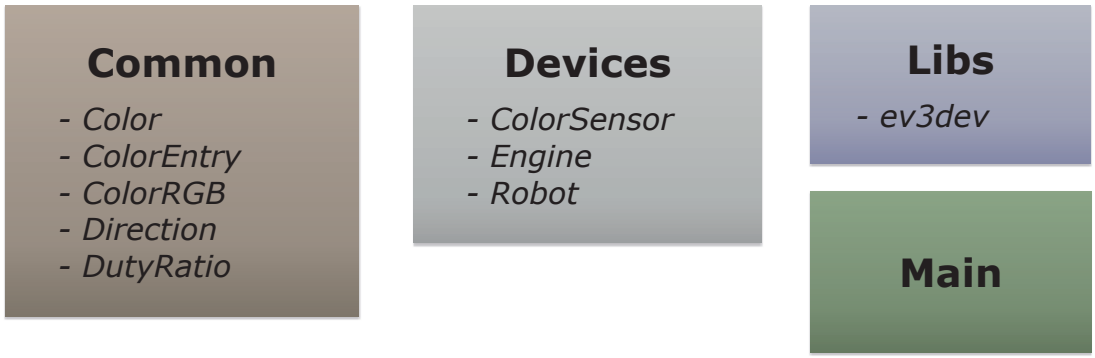
Considérations techniques :

*Nous avons installés le système **EV3Dev** de **ev3dev.org** sur la brique Lego EV3. C'est un Linux basé sur **Debian** ce qui nous permet de communiquer facilement en ssh avec le robot. De plus plusieurs librairies sont disponibles dans divers langage de programmation. Celle qui paraît la plus aboutie est en **C++**, ce qui est une des raisons de notre choix pour ce langage. Nous l'avons choisis également pour sa rapidité d'exécution et son absence de machine virtuelle étant donné que l'**EV3** n'a que **60Mo** de **RAM**. Ce qui rend d'ailleurs le python impraticable comme nous l'avons constatés. De plus nous faisons de la cross-compilation par soucis de rapidité. L'architecture cible est une **ARM**, plusieurs compilateurs sont disponibles, un lien est disponible dans le **README.md** (Mac - Windows).*

Architecture et décomposition :

*Tout d'abord nous avons un **Makefile** contenant une règle **make send** qui envoi l'exécutable sur le robot en ssh. Il y a deux scripts bash dans le dossier **tools/** qui permettent d'installer ou de désinstaller les outils de développement nécessaires. Typiquement la chaîne de développement est très courte : modification du code, cross-compilation avec **make**, push du nouvel exécutable avec **make send**. Les modifications sont donc faciles et rapide a effectuer.*

La répartition des fichiers est la suivante :



Une seule librairie extérieur est donc utilisée, elle permet la communication facile avec les capteurs et moteurs grâce à une série d'objets et de fonctions préconçues.
*Dans le dossier **Common/** se trouvent les différentes structures de données et énumérations. **Devices/** est la partie contenant les modèles, le **Robot.h** contient une instance de **ColorSensor** et une instance de **Engine**. Enfin le point de départ est **Main.cpp**.*

Cette architecture permet un découpage qui se rapproche au plus prêt des objets réels. Les algorithmes sont plus intuitifs avec une telle approche, et facilement modulable. Si nous voulions rajouter un capteur, il suffirait d'ajouter un objet ColorSensor au robot.

Capteur de couleur : calibration et lecture

Calibration :

La calibration permet d'enregistrer les résultats de lecture d'une charte colorimétrique dans un fichier. Ce fichier sera utilisé par le capteur pour lire une couleur.
*L'opération se déroule de la façon suivante : une lecture de **50 échantillons** est réalisée de manière à sauvegarder les composantes **RVB Max** et **RVB Min**.*

Plus précisément un triplet contenant tout les maximums des trois composantes, et de façon analogue pour les minimums :

Valeur max = ColorRGB (max R, max G, max B)
Valeur min = ColorRGB (min R, min G, min B)

On obtient donc une «fenêtre» contenant la couleur lu et ses variations.

Lecture :

*Lors de la lecture, on fait une moyenne sur **10 échantillons** et on compare le triplet RVB à ceux contenu dans le dictionnaire, préalablement construit grâce au fichier de calibration. En comparant la distance entre l'échantillon et le point Max et Min de chaque entrée du dictionnaire, on peut retrouver la couleur qui s'en rapproche le plus.*

$$c1 = RVB(140,\ 140,\ 140)$$
$$c2 = RVB(150,\ 160,\ 160)$$

$$\mathbf{distance(c1,\ c2)} = \sqrt{(R_{c1} - R_{c2})^2 + (V_{c1} - V_{c2})^2 + (B_{c1} - B_{c2})^2}$$

Propulsion et contrôles

Gestion des moteurs :

*Les deux moteurs sont contenu dans l'objet **Engine** qui dispose des fonctions de contrôle de direction, de vitesse et d'arrêt. Les algorithmes des fonctions du **Robot** utilisent donc celles-ci pour manœuvrer.*