.

# Projet Robot LEGO : Cahier des charges. Cours Implantation de systèmes de commande embarquée

Thao Dang et Pascal Raymond

10 octobre 2017

Le but de ce projet est de faire suivre à un robot LEGO une ligne noire sur le sol tout en évitant des obstacles qui éventuellement apparaissent devant le robot.

Le projet se fait en 2 étapes principales : conception d'un contrôleur, et implantation du contrôleur. En particulier, nous allons appliquer deux méthodes d'implantation : mono-tâche et multi-tâche. Afin d'illustrer la première, nous considérons seulement l'objectif de suivre la ligne, et afin d'illustrer la deuxième, nous ajoutons l'objectif d'évitement d'obstacles.

## 1 Implantation mono-tâche - suiveur de ligne

### 1.1 Conception du contrôleur de suivi de ligne

Afin de suivre la ligne sur le sol, le robot utilise deux capteurs de lumière $G$ (gauche) et $D$ (droite) qui sont positionnés sur le devant et pointent vers le sol afin de détecter la ligne noire. Le robot possède deux moteurs connectés à ses deux roues dont les vitesses sont $v_G$ et $v_D$. Les capteurs sont utilisés pour estimer la déviation de la trajectoire du robot par rapport à la ligne noire.

Un contrôleur est proposé et implanté dans un modèle Simulink disponible sur la page du cours. Il est important de noter que ce contrôleur est donné à

titre d'exemple. Pour l'adapter à des environnements concrets, il faut modifier ses paramètres.

Le contrôleur de suivi de ligne se compose de deux blocs :

— Bloc de calcul d'écart. Ce bloc calcule les erreurs d'orientation et de vitesse. Les capteurs sont utilisés pour mesurer l'écart de la trajectoire actuelle du robot par rapport à la ligne noire, à partir des données $c_G$ et $c_D$ fournies par les capteurs à gauche et à droite.

— Bloc de correction. Les erreurs d'orientation et de vitesse sont les entrées du bloc de correction. L'orientation est corrigée par un correcteur de type PI, et l'erreur de vitesse par un correcteur de type P.

  Notons que la vitesse du robot dépend également de l'erreur d'orientation, puisque quand l'erreur d'orientation à corriger est grande (c-à-d le robot n'est pas du tout aligné à la ligne), il faut ralentir le robot afin que le robot ne quitte pas la ligne. **Question : quels blocs dans le contrôleur réalisent cette régulation ?**

**À faire :** Ajuster les paramètres du contrôleur et valider son fonctionnement et sa performance via le simulateur de Simulink.

## 1.2 Implantation du contrôleur de suivi de ligne

### 1.2.1 Échantillonnage

Il s'agit de discrétiser le contrôleur avec une période d'échantillonnage fixe.

**Question** : expliquer le choix de la période d'échantillonnage et de la méthode de discrétisation (Euler en avant, Euler en arrière, ou autre).

### 1.2.2 Calibrage des moteurs

**À faire :**
Si l'on spécifie les vitesses de consigne pour les deux roues par le pourcentage par rapport à la puissance maximale, il faut déterminer pour chaque robot concret les vitesses maximales que ses deux moteurs peuvent produire. Ces valeurs dépendent évidemment du niveau de la batterie.

De plus, il se peut que les deux moteurs ne fonctionnent pas exactement de la même manière. Il faut alors corriger les commandes moteur de manière à ce qu'une consigne identique sur les deux roues fasse bien rouler le robot tout droit.

**Question** : Expliquer le calibrage des moteurs.

### 1.2.3 Calibrage des capteurs

Le contrôleur en Simulink fourni a été développé sous l'hypothèse que l'intervalle de valeurs données par des capteurs de lumière va de 0 (noir) à 100 (blanc).
**Question** : est-ce que cette hypothèse est valide pour les capteurs réels ? Sinon, il faut effectuer un calibrage. Expliquer le calibrage.

### 1.2.4 Génération de code

On utilise la chaîne **mdl2lus2osek** pour générer le code du contrôleur et l'exécuter sur le robot en utilisant la méthode mono-tâche :

- Le contrôleur Simulink est traduit en programme Lustre, puis le programme Lustre est lui-même traduit en programme C.
- Seul le code purement fonctionnel est géné automatiquement : tout ce qui concerne les entrées-sorties de la briques doit être programmé par directement en C (c'est ce qu'on appelle le code *glue*). Voir l'API de nxtOSEK [1] pour les fonctions de manipulation de capteurs et de moteurs. Le code de glue doit contenir dans la phase d'initialisation (par exemple dans la fonction "`usr_init`") une étape de "calibrage" de capteurs et de moteurs.
- Le contrôleur C, le code glue et les librairies système sont compilées et assemblées pour obtenir le fichier binaire, qu'on peut alors charger et exécuter sur la brique.
- Un fichier Makefile de base, qu'il faut adapter, est fourni pour enchaîner toutes les phases de compilation.
- Une page spécifique avec toutes les informations nécessaires est diponible sur le site du cours.

1. `http://lejos-osek.sourceforge.net/ecrobot_c_api.htm`

## 1.3 Expérimentation

**À faire :** tester le code généré sur le robot et débogger si besoin.

**Question** : expliquer et interpréter la procédure d'expérimentation et les résultats.

# 2 Implantation multi-tâche - suiveur de ligne et anti-collision

## 2.1 Conception d'un planificateur

**À faire :** Nous allons créer un planificateur qui assure et coordonne les deux fonctions correspondant aux deux objectifs de commande :

- Correction de trajectoire. Nous allons réutiliser le correcteur de trajectoire de la section précédente. Ce correcteur assure que le robot corrige toute erreur par rapport à une trajectoire de référence. Dans le mode suiveur de ligne, cette trajectoire de référence est la ligne.
- Anti-collision. rencontre un obstacle, le planificateur génère une commande de changement de direction et/ou de vitesse (par exemple, arrêt, avancement en vitesse réduite) quand le robot rencontre un obstacle. Le planificateur détermine d'une manière continue les valeurs de consigne $\epsilon_d$ et $\epsilon_\theta$ et puis communique ces valeurs aux contrôleurs de trajectoires ci-dessus.

  La *stratégie d'anti-collision* que l'on va utiliser est la suivante ; sur détection d'un obstacle :
  - d'abord, le robot s'arrête (notons que le robot prend un certain temps pour s'arrêter complètement !).
  - ensuite, le robot fait un demi-tour sur place. Supposons que le robot tourne vers le capteur à gauche, pour détecter le moment où le robot termine le demi-tour, on peut détecter une séquence de valeurs « Blanc-Noir-Blanc » sur la sortie du capteur gauche. Une méthode de détection est décrite dans la suite.

**Planificateur**

- Dans le mode suiveur de ligne, le planificateur détermine d'une manière continue les valeurs de consigne $\epsilon_d$ et $\epsilon_\theta$, puis communique ces valeurs aux contrôleurs (de distance et d'angle) qui calculent les vitesses des deux roues.
- Dans le mode anti-collision, il est important que le robot s'arrête le plus rapidement possible : *il est donc plus efficace pour le planificateur de spécifier directement les vitesses $v_d$ et $v_g$ des deux roues*, au lieu de spécifier les erreurs $\epsilon_d$ et $\epsilon_\theta$ à corriger.

On peut voir que quand il y a seulement le mode suiveur de ligne (comme dans la section précédente, la planification est assurée par le bloc Calcul d'écart. Maintenant, pour gérer les deux modes, il faut donc modifier le bloc Calcul d'écart en y ajoutant la gestion d'évitement d'obstacles.

**Détecter une séquence de couleurs.** Pour détecter une séquence de valeurs « Blanc-Noir-Blanc », il faut mémoriser, par exemple, l'événement qu'un capteur voit la couleur « Blanc ». Pour ceci, on peut utiliser un bloc *Logical Or* dont une entrée est connectée à la sortie d'un comparateur (voir la Figure 1 où $N_b$ est le seuil de la couleur blanche).

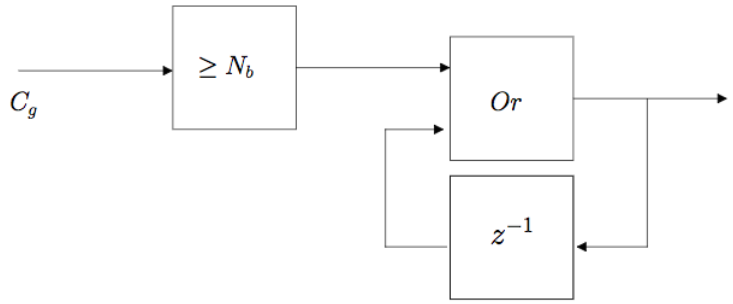


FIGURE 1 – Mémorisation de l'événement de détection de couleur blanche.

Après avoir terminé le demi-tour, le système de commande doir revenir dans l'état avant le mode d'anti-collision, il est utile de pouvoir *réinitialiser* les variables importantes du système. Pour faire ceci, voir la Figure 2 pour un exemple de bloc *mémoire avec reset*.

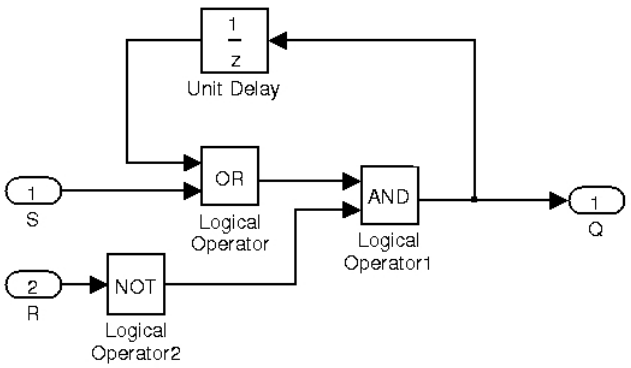


FIGURE 2 – Mémoire avec reset.

## 2.2 Implantation multi-tâche

On utilise la chaîne **mdl2lus2osek** pour générer le code du contrôleur et l'exécuter sur les robots en utilisant deux méthodes : mono-tâche et multi-tâche.

- Mono-tâche : similaire à ce qu'on a fait en section 1.
- Multi-tâche : on sépare le contrôleur en deux parties, qui seront chacune exécutée par deux tâches système périodiques. Cette expérience a pour but nottamment d'illustrer les problèmes de communication et d'interférence dans les implémentations multi-tâches préemptives. Comme dans la section 1, on dispose d'un Makefile de base à adapter et compléter pour enchaîner les phases de compilation.

**À faire :**

1. Pour l'implantation multi-tâche, séparer le contrôleur en deux sous-systèmes représentant deux tâches avec différentes périodes, puis générer le code Lustre (en utilisant l'outil **mdl2lus**), et le code C (en utilisant le compilateur **lus2c**) pour chaque sous-système. Expliquer la séparation.
2. Faire des expériments sur le robot et ajuster la commande si besoin en modifiant le modèle Simulink. Expliquer la procédure et interpréter le résultat.