
# Système temps-réel, modélisation réaliste et implantation

Julien BONNARDEL

&

Hadrien de KERGORLAY

8 février 2017

# Table des matières

# 1 Introduction

Ce projet a pour but d'implémenter deux fonctions sur le Robot Lego : Suivre une ligne noire et éviter un obstacle. Pour éviter un obstacle, le robot doit faire demi tour et repartir dans l'autre sens, en suivant la piste.

Nous devons au cours de ce projet, modéliser, simuler et valider. Ce sont les étapes à faire avant d'implémenter et d'exécuter une fonction dans un système critique temps réel.

C'est l'industrie de contrôle de sécurité des systèmes critiques temps réels qui est à l'origine d'une méthode de développement d'un modèle très rigoureuse. L'implantation des systèmes de contrôle est dite sure lorsqu'elle respecte deux propriétés : temps réel et déterministe. Déterministe signifie qu'on doit connaître le résultat d'une simulation. Le but de ce projet est en fait de découvrir les méthodes classiques de conception et d'implantation sures.

L'an dernier, nous avions découvert l'implantation synchrone d'un système échantillonné et mono-tache. Nous avions revu l'intérêt d'avoir un système stable et certains critères qui permettaient de déduire la stabilité de notre système. Cette année, nous voyons l'implantation d'un système multi-taches et ferons une analyse de la robustesse de notre système.

Cette notion de robustesse est nouvelle et sera présentée dans la section suivante.

# 2 Modélisation réaliste et commande robuste

La commande robuste vise à déterminer une loi de commande qui soit capable de garantir des critères de performance et stabilité pour un système dont le modèle varie autour du modèle théorique ou nominal. En effet, le modèle mathématique qui modélise un système réel est une représentation qui vise à approximer au mieux, avec des hypothèses simplificatrices, le système qu'on veut commander. Il existe donc un écart entre le comportement observé du système réel et son modèle interne. Par la même approche, on peut rendre le système robuste face aux perturbations extérieures. On dit qu'un système est robuste si la régulation fonctionne toujours même si le modèle change un peu. Le système doit rester stable en considérant certaines imprécisions ou perturbations.

Nous devons donc évaluer la robustesse de notre modèle. Notamment la robustesse du contrôleur d'orientation. Un contrôleur parfait à un temps de réponse rapide et converge vers le signal d'entrée. Nous connaissons le compromis entre performances et stabilité : un système plus rapide risquera d'être instable. Nous devons nous assurer que notre modèle restera stable pour toutes les valeurs admissibles de l'incertitude. On doit pour cela étudier la réaction de notre modèle à des perturbations.

## 2.1 Modélisation des sources d'incertitudes

La première étape pour assurer la robustesse est d'envisager correctement les sources d'incertitudes du modèle.

Plusieurs sources d'imprécisions sont possibles. Il y a d'abord les perturbations externes, les imprécisions des capteurs, mais aussi des imprécisions liées à notre modèle dues à la discrétisation, aux erreurs numériques et à la non linéarité. Ces sources d'incertitudes peuvent faire varier notre modèle autour de son fonctionnement nominal.

Comme on l'a mentionné précédemment, la discrétisation peut être une source d'incertitude. L'an dernier, nous avions vu trois méthodes permettant de passer d'un système continu à un système numérique. Pour cela, nous devions choisir une fréquence d'échantillonnage et une

méthode. La méthode de Tustin s'avérait la plus efficace car permet de conserver la stabilité dans l'espace des Z. Cela signifie que les pôles en Z du polynôme caractéristique doivent être dans le cercle unitaire. Sinon, le système est instable.

Ces imprécisions ont une influence sur la stabilité du contrôleur. Un contrôleur instable ne pourra pas converger vers le signal d'entrée. Il oscillera autour de ce signal. En lançant une simulation après avoir ajouté une perturbation, on s'aperçoit que le robot suit la ligne par à-coups successifs. Il tournera en permanence en alternant droite gauche. Cela signifie que notre système global est instable.

Nous allons brièvement présenter le robot avant d'étudier la robustesse du contrôleur d'orientation du robot.

## 2.2 Présentation du robot

Le robot est constitué de deux capteurs de lumière situés à l'avant et sur les cotés. Lorsqu'un capteur voit du noir, il renvoie une valeur proche de 0. S'il voit du blanc, il renvoie une valeur proche de 100. Le calibrage des capteurs afin que la valeur renvoyée se situe entre 0 et 100 a été effectué l'an dernier.

Un nouveau capteur de distance permet de détecter les obstacles. Pour commander le robot, deux contrôleurs distincts sont implantés : un contrôle commandant l'orientation du robot, et un contrôleur commandant la vitesse du robot. Et nous nous intéressons à présent à la stabilité du contrôleur d'orientation. Pour cela, on utilisera les critères de stabilité pour des systèmes incertains et avec retard. Ces critères sont les critères d'Hurwitz et de Bialas et seront détaillés plus tard.

## 2.3 Le contrôleur d'orientation

On va expliquer le fonctionnement du contrôleur d'orientation. On aboutira aux équations différentielles qui nous permettrons, après traduction en Laplace, d'obtenir une fonction de transfert, On ajoutera un correcteur dans le schéma bloc. Le rôle d'un correcteur étant de corriger le système afin que la réponse du système soit performante. Les trois critères de performances sont la rapidité, la précision, et la stabilité. Nous calculerons ensuite la fonction de transfert en boucle fermée pour en déduire la stabilité de notre système en fonctionnement nominal. Cette stabilité sera vérifiée par des critères déjà bien connus.

Une fois la stabilité en fonctionnement nominale attestée, nous mettons en oeuvre ce que nous avons appris lors de ce cours afin d'analyser la robustesse de ce contrôleur vis à vis de trois paramètres susceptibles de varier. Ces paramètres sont T la période d'échantillonnage, $k_{i\theta}$ le coefficient de l'action intégrale et $k_{p\theta}$ le coefficient de l'action proportionnelle.

Pour cela, on calculera la matrice d'Hurwitz et on appliquera le critère de Bialas pour déduire la robustesse de notre système.

## 2.4 Fonction de transfert

On modélise la trajectoire du robot de la façon suivante : Vd et Vg sont les vitesses des roues droite et gauche. A partir des équations différentielles, nous obtenons la formule de l'orientation du robot :

$$\theta = \frac{v_g - v_d}{l}$$

Nous contrôlons l'orientation du robot par un contrôleur de type PI(proportionnel intégral) dont la fonction de transfert est :

$$C_\theta(\mathrm{s}) = \frac{k_{i\theta}}{s} + k_{p\theta}$$

où $k_{i\theta}$ est le coefficient de l'action intégrale et $k_{p\theta}$ le coefficient de l'action proportionnelle.

Conformément à l'équation donné par la physique, nous utilisons la différence de vitesses des roues pour orienter le robot. La variable de commande $u_\theta$ vaut :

$$u_\theta = v_g \text{ - } v_d$$

On calcule alors la fonction de transfert, puis la fonction de transfert en boucle fermée qui nous permettra d'évaluer la stabilité de notre système. On obtient un système du second ordre dont les pôles doivent être à partie réelle négative afin d'assurer la stabilité.

On pose $\alpha = 1/l$.

On obtient alors :

$$k_{i\theta} = \omega^2/\alpha \text{ et } k_{p\theta} = 2\xi\omega/\alpha$$

$\xi$ étant l'amortissement.

### 2.5 Rappel sur la stabilité

Les 2 poles du système en boucle fermée sont :

$$p_1 = \text{-}\omega(\xi + i\sqrt{1-}\xi^2) \; et \; \mathrm{p}_2 = -\omega(\xi - i\sqrt{1-}\xi^2)$$

Les deux pôles sont à partie réelle négative ce qui garantit la stabilité. De façon générale, le polynôme caractéristique doit avoir toutes ses racines dans le demi-plan gauche complexe en continu. Après avoir discrétisé, la zone spécifique de stabilité est le disque unité du plan complexe.

$\omega$ et $\xi$ doivent être choisis de sorte à avoir les performances désirées. Le temps de réponse à 5%, le dépassement, la précision sont des critères de performances.

### 2.6 Critères de robustesse

Les outils d'analyse de robustesse seront exprimés. Des méthodes mathématiques permettent en effet de garantir ou non la robustesse d'un système. Nous étudions le cas de notre contrôleur d'orientation. Nous étudions le cas où seul un paramètre est incertain et son intervalle d'incertitude est connu.

On rappelle que la fonction de transfert en boucle fermée est :

$$H = \frac{\frac{k_{p\theta}}{l}s + \frac{k_{i\theta}}{l}}{s^2 + \frac{k_{p\theta}}{l}s + \frac{k_{i\theta}}{l}}$$

On discrétise par la méthode de Tustin. Cela revient à remplacer s par la fonction de z suivante :

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

$T$ étant la période d'échantillonnage.

Afin d'utiliser les mêmes critères que pour un système en temps continu, on va appliquer la transformation en Z par le changement de variable suivant :

$$z = \frac{1+\eta}{1-\eta}$$

Ce changement de variable en $\eta$ permet de ramener le cercle unitaire en z vers le demi-plan gauche en  (les pôles du polynôme caractéristique doivent être dans cette zone pour garantir la stabilité).

Nous pouvons maintenant traiter la variable $\eta$ comme la variable s dans les fonctions de transfert en Laplace.

Finalement, le polynôme caractéristique devient :

ôme.png ôme.png ôme.pdf ôme.pdf ôme.jpg ôme.jpg ôme.mps ôme.mps ôme.jpeg ôme.jpeg ôme.jbig2 ôme.jbig

### 2.6.1 Critère des valeurs propres

Par un script fourni, on calcule la matrice de Hurwitz pour déterminer la stabilité du polynôme caractéristique pour différentes valeurs de la période d'échantillonnage T et des coefficients $k_{p\theta}$ et $k_{i\theta}$ du correcteur PI. On rappelle que la matrice de Hurwitz est calculée en fonction des coefficients du polynôme caractéristique et de son degré. Si tous les mineurs principaux dominants de la matrice sont positifs, alors le système est stable.

Les racines dépendent d'une manière continue des coefficients du polynôme, donc l'instabilité surgit quand les racines traversent la frontière de stabilité (l'axe imaginaire dans le cas de temps continu).

Pour différentes valeurs de la fréquence d'échantillonage, de $k_{p\theta}$ et $k_{i\theta}$, on calcule la matrice de Hurwitz et on vérifie si le critère de stabilité est respecté. On s'assure donc de choisir des valeurs pour ces coefficients tel que le critère des valeurs propres soit bien respecté.

### 2.6.2 Critère de Bialas

Le but du critère de Bialas est de trouver les conditions sur les paramètres afin d'assurer la stabilité robuste du système. On écrit le polynôme caractéristique sous la forme suivante : $\pi(\eta) = \pi_0(\eta) + q\pi_1(\eta)$ où $\pi_0$ est stable et de degré supérieur à $\pi_1$. q est le paramètre incertain de notre système.

On identifie $\pi_0(\eta), q, \pi_1(\eta)$ à partir du polynôme caractéristique précédent. q dépend de T, $k_{p\theta}$ et $k_{i\theta}$. On fait à présent varier le paramètre q en s'assurant à chaque fois que le polynôme reste stable. On détermine ainsi la marge de robustesse de notre système.

### 2.6.3 Stabilité pour un système incertain avec retard

On se souvient quand Laplace, f(t-$\tau$) se traduit par F(s)*exp(-$\tau$s). Le délai du moteur peut être introduit dans la fonction de transfert de la façon suivante :

$$H_\theta(s) = \frac{exp(-\tau s)}{ls}$$

Ce qui donne une nouvelle fonction de transfert dont on calcule le nouveau polynôme caractéristique. On applique ensuite la même méthode et les mêmes critères de stabilité pour en déduire l'influence d'un retard sur la stabilité de notre système.

# 3 Implémentation simulink du système anti-collision

## 3.1 Module anti-collision - suiveur de ligne

Une fois la théorie assimilée, on crée un modèle simulink permettant de simuler le comportement de notre robot face à un obstacle. On choisit tout d'abord une stratégie d'évitement d'un obstacle :

— Le robot s'arrête (pas instantanément du fait de la latence des moteurs).
— Puis il fait un demi tour.

Pour conserver le caractère de suiveur de ligne, il faut intégrer la capacité au robot de reprendre la ligne. On suppose que le robot fait un demi tour en partant sur sa gauche lorsqu'il rencontre un obstacle. Autrement dit le capteur gauche du robot va observer une séquence "Blanc-Noir-Blanc". Le robot a donc besoin de mémoriser la séquence de couleur. Pour ce faire on utilise un bloc de mémorisation avec reset qui permet au contrôleur de revenir dans l'état antérieur au mode anti-collision.

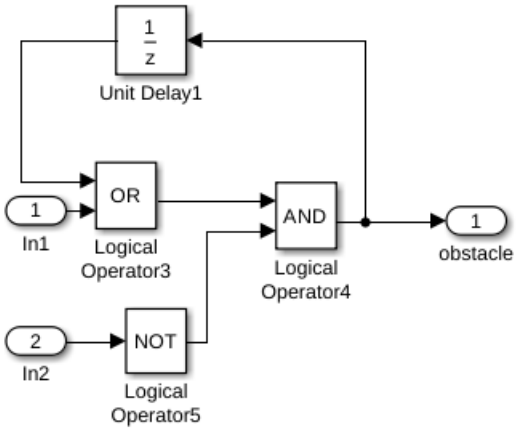


FIGURE 1 – Bloc mémoire avec reset.

Puis on crée le sous-système qui permet de reconnaître la séquence "Blanc-Noir-Blanc" et de savoir si il est dans le mode suiveur de ligne ou dans le mode détection d'un obstacle (demi tour).

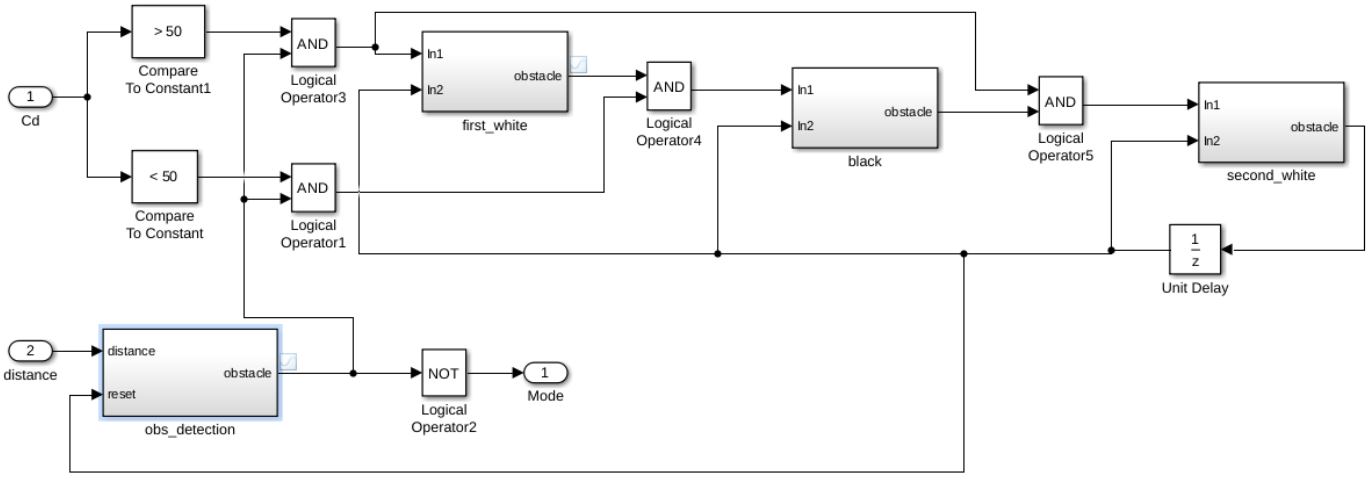


FIGURE 2 – Bloc de changement de mode.

On a rencontré quelques soucis lors de la simulation de ce bloc, on a donc rajouté un bloc de détection d'obstacle (obs_detection) qui nous permet de savoir si oui ou non il y a un obstacle à proximité. On test la distance (valeur reçu par le capteur) avec une distance minimal fixée arbitrairement. En sortie du bloc obs_detection, on a soit oui il y a un obstacle (signal à 1), soit non il y a pas d'obstacle (signal à 0).

## 3.2 Module du comportement

Une fois le module d'anti-collision et suiveur de ligne mise au point, il faut modifier le comportement du robot en fonction. Lorsque le robot détecte un obstacle, il effectue un demi tour. Sinon il reste dans sa position de suiveur de ligne. On a donc ajouté un petit sous-système qui permet au robot d'inverser le sens de rotation de d'un de ces moteurs (passer en marche arrière).

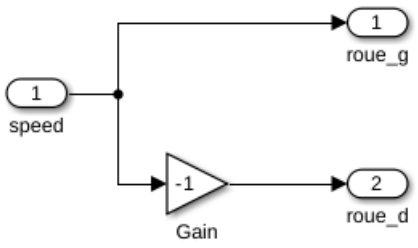


FIGURE 3 – Bloc de demi tour.

Le bloc du comportement du robot était quasiment implémenté, il a juste fallu rajouter les switch du faite des deux modes possibles (suiveur de ligne, anti-collision). Le choix de la vitesse à 0.4 est purement arbitraire. Ceci permet au robot de faire son demi tour à vitesse correcte.

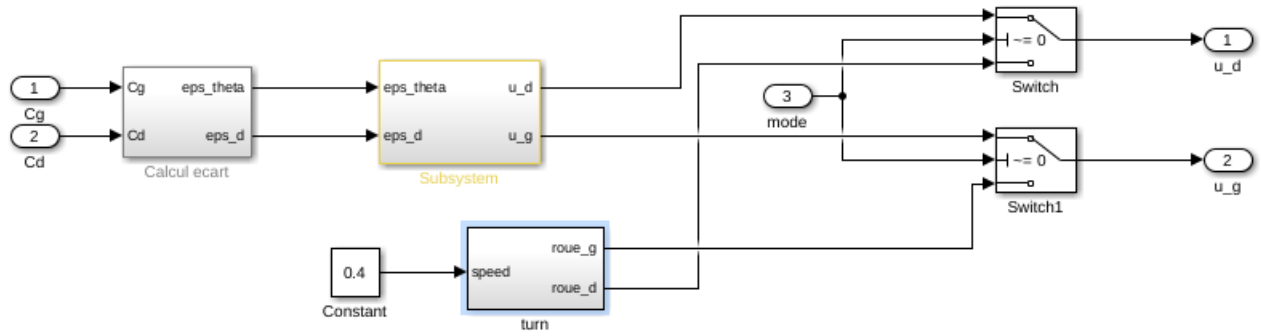


FIGURE 4 – Bloc de comportement.

### 3.3 Le système de contrôle anti-collision - suiveur de ligne

Voici donc le système de contrôle anti-collision et suiveur de ligne du robot Lego :

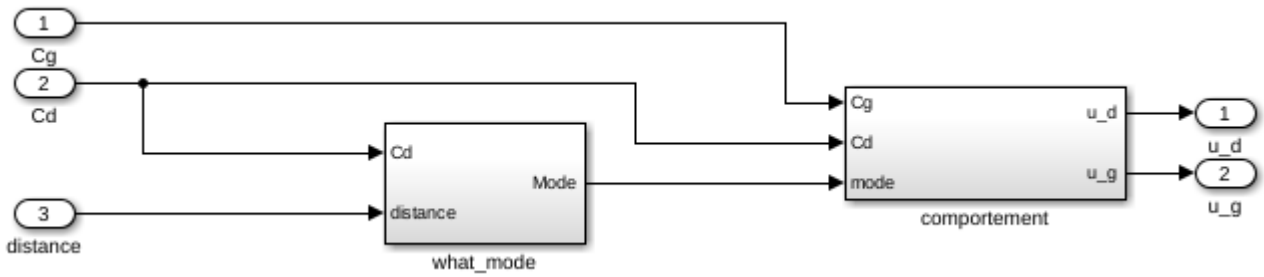


FIGURE 5 – Système de contrôle.

Il se compose donc du bloc "what_mode" qui sert à savoir si le robot est en suiveur de ligne ou traite le cas de l'évitement d'un obstacle. Lorsque ce mode est connu, il envoie ces information au bloc "comportement" qui s'occupe soit de faire un demi tour, soit de suivre la ligne.

Pour valider le tout, on simule l'obstacle au sein de simulink à l'aide d'un générateur de créneau "simu_obstacle" (il y a un "obstacle" tout les trois secondes).

## 4 Implémentation système anti-collision

### 4.1 Mise place du cadre de travail

Une fois la simulation validée par simulink, il faut générer les différents exécutables et compléter le code c (glue.c) pour que le robot soit fonctionnel. Le modèle respecte bien la séparation du contrôleur en deux sous-systèmes (cf. Figure 5) "what_mode" et "comportement".

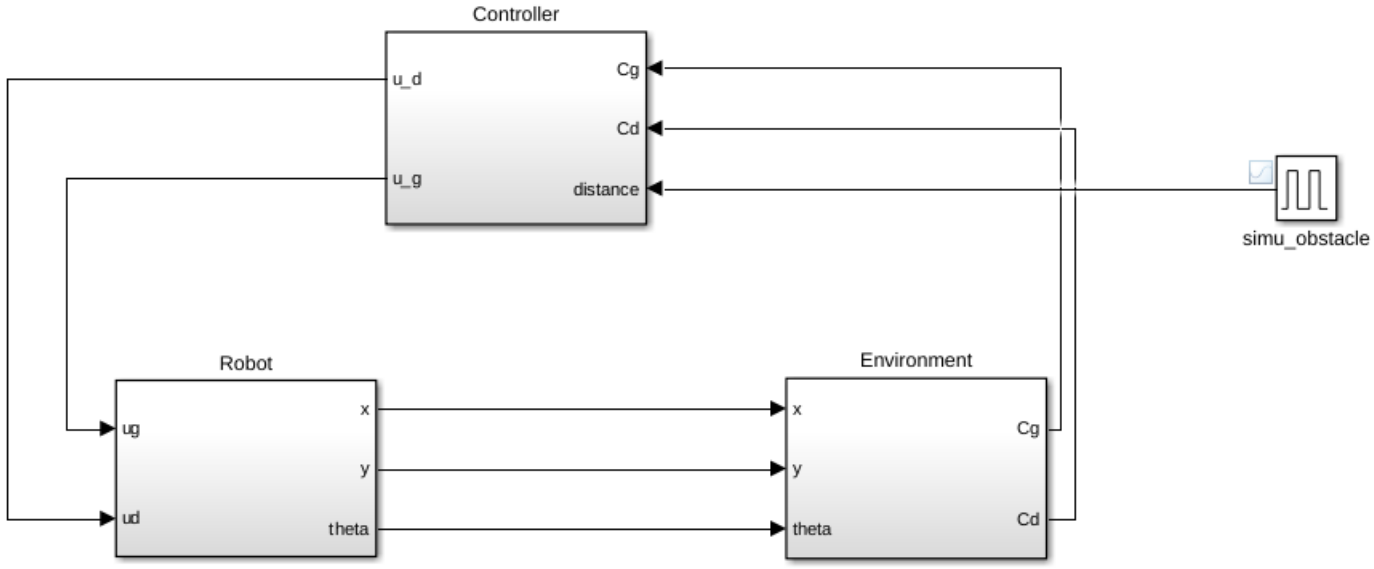


FIGURE 6 – Environnement du robot.

Pour générer les fichiers binaires, on respecte les schéma suivant :

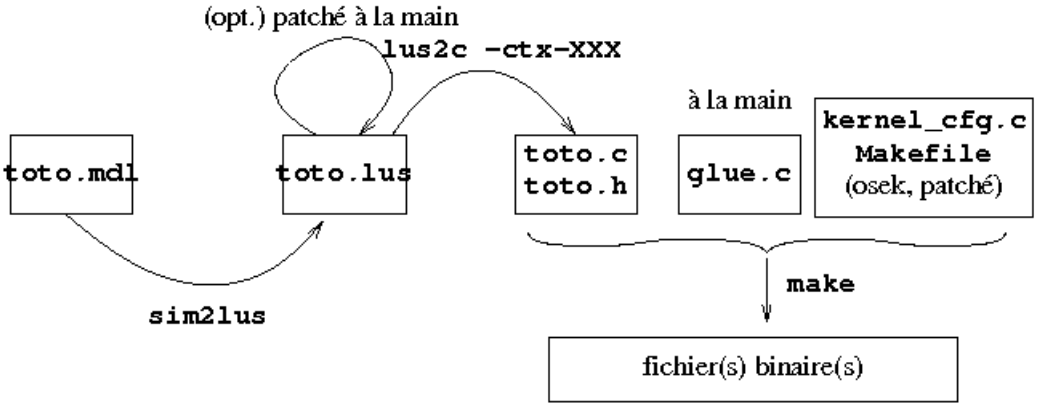


FIGURE 7 – Chaine de compilation.

On crée les fichiers lus des deux sous-système, "what_mode.lus" et "comportement.lus" en tapant la commande "mdl2lus ../ExRobotAndEnvironmentController.mdl -system comportement (et what_mode)".

Puis on génére les fichiers c et h de ces derniers, "what_mode.c what_mode.h" et "comportement.c comportement.h" en tapant la commande "lus2c comportement.lus comportement -ctx-static (et what_mode)".

De plus on importe les fichiers kernel_cfg.c, kernel_id.h et un Makefile que l'on complète comme ceci :

Et on édite deux fichiers "comporetement_ext.h" et "what_mode_ext.h" où l'on définit des constantes pour le bon fonctionnement du robot.

```
:
# Application name
TARGET=MyProgram

# Where to find user C code
USR_PATH=.

# -----------
# user C code
# -----------
USR_CFILES=\
   kernel_cfg.c\
   what_mode.c\
   comportement.c \
   glue.c\

include $(MDL2LUS2OSEK)/generic.mak
```

FIGURE 8 – Makefile.

## 4.2 Implémentation du code c (glue.c)

Le cadre de travail étant fait. Il suffit de compléter le code du fichier glue.c en utilisant l'API nxtOSEK qui se trouve sur le site "http ://lejos-osek.sourceforge.net/ecrobot_c_api.htm".

— Initialisation et désactivation des capteurs :

```
/* Init and terminate OSEK */
void ecrobot_device_initialize() {
   /*
      HERE: put here specific code that will be executed ONCE
            when the application starts
      TYPICALLY: initialization of (light) sensors
   */
        ecrobot_init_sonar_sensor(PORT_C_DISTANCE);
        ecrobot_set_light_sensor_active(PORT_C_GAUCHE);
        ecrobot_set_light_sensor_active(PORT_C_DROITE);
}

void ecrobot_device_terminate() {
   /*
      HERE: put here specific code that will be executed ONCE
            when the application stops
      TYPICALLY: finalization of (light) sensors
   */
        ecrobot_term_sonar_sensor(PORT_C_DISTANCE);
        ecrobot_set_light_sensor_inactive(PORT_C_GAUCHE);
        ecrobot_set_light_sensor_inactive(PORT_C_DROITE);
}
```

FIGURE 9 – Initialisation et désactivation des capteurs.

— Initialisation du mode et calibrage des capteurs :
— Implémentation de la tâche de basse priorité (sous-système what_mode) :

— Implémentation de la tâche de haute priorité (sous-système comportement) :
— Implémentation des fonctions de sorties de vitesse moteur en fonction de la sortie du système de contrôle :

Puis il reste à compiler le tout grâce au Makefile en tapant "make", de connecter le robot par USB au pc et taper la commande "t2n -put MyProgram.rxe" pour envoyer le code dans le robot.

```
void usr_init(){
        /*Initialisation du buffer pour le mode*/
        buf_what_mode[0] = true;
        buf_what_mode[1] = true;

        GetResource(lcd) ;

        display_clear(0);
        display_goto_xy(0, 0);
        display_string("Mettre le robot");
        display_goto_xy(0, 1);
        display_string("sur le blanc");

// TO DO
        while(!ecrobot_is_ENTER_button_pressed()) {

                blanc_gauche = ecrobot_get_light_sensor(PORT_C_GAUCHE) + 20;
                blanc_droite = ecrobot_get_light_sensor(PORT_C_DROITE) + 20;

                display_clear(1);
                display_goto_xy(0, 0);

                display_string("---- Blanc ----");

                display_goto_xy(0, 1);

                display_string("G => ");

                display_goto_xy(5, 1);
                display_int(blanc_gauche, 3);
                display_goto_xy(0, 2);

                display_string("D => ");
```

FIGURE 10 – Début de la fonction de calibrage.

```
TASK(LowTask) { // ==> What_mode

        display_goto_xy(0, 0);
        display_string("What mode !");
        display_update();

        //Valeur des capteurs
        what_mode_I_distance(distance_check(ecrobot_get_sonar_sensor(PORT_C_DISTANCE)));
        what_mode_I_Cd(calib(ecrobot_get_light_sensor(PORT_C_DROITE),noir_droite, blanc_droite));

        what_mode_step();

        TerminateTask();
}
```

FIGURE 11 – Tâche de basse priorité.

```
TASK(HighTask) { // ==> Comportement

        if (compteur == 0){
                actual_mode = (actual_mode + 1) % 2;
        }

        display_goto_xy(0, 0);

        if (buf_what_mode[actual_mode % 2]) {

                display_clear(1);
                display_goto_xy(1, 6);
                display_string("J'avance");

        } else {

                display_clear(1);
                display_goto_xy(1, 3);
                display_string("-- Obstacle --");
                display_goto_xy(1, 7);
                display_string("Je tourne !");

        }

        display_update();
```

FIGURE 12 – Début de la tâche de haute priorité.

```
void comportement_O_u_d(_real vit) {

        int vit_int = (int) (vit * 100);

        if (vit_int > VMAX) {
                vit_int = VMAX;
        }
        if (vit_int < -VMAX) {
                vit_int = -VMAX;
        }

        display_goto_xy(0, 1);
        display_int(vit_int, 3);
        display_update();

        ecrobot_set_motor_speed(NXT_PORT_A, vit_int);
}
```

FIGURE 13 – Vitesse moteur droit.

```
void comportement_O_u_g(_real vit) {

        int vit_int = (int) (vit * 100);

        if (vit_int > VMAX) {
                vit_int = VMAX;
        }
        if (vit_int < -VMAX) {
                vit_int = -VMAX;
        }

        display_goto_xy(0, 2);
        display_int(vit_int, 3);
        display_update();

        ecrobot_set_motor_speed(NXT_PORT_B, vit_int);
}
```

FIGURE 14 – Vitesse moteur gauche.

# 5 Conclusion

Par rapport à l'an dernier, deux éléments majeurs ont été étudié : la robustesse et l'implantation multi-taches. Nous avons envisagé un modèle plus réaliste, qui prend en compte l'incertitude de certains paramètres. Notre système reste-t-il stable ? Pour répondre à cette question, il faut faire une étude de la robustesse de notre système.

Enfin, la détection d'obstacles qu'on a ajouté a augmenté la complexité de notre système en le rendant multi-taches. Pour répondre aux problématiques temps réels, il fallait décomposer le contrôleur en deux sous-systèmes. Un contrôleur de planification se charge de définir quelle est la tache prioritaire et quelle tache doit être exécutée.