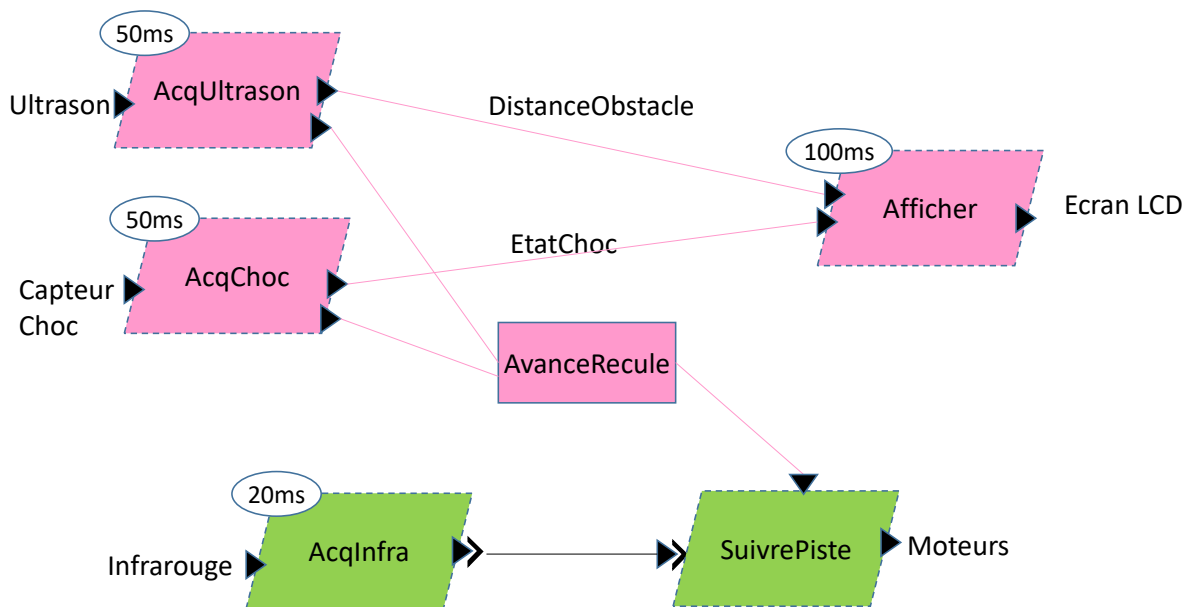


Programmation d'un robot NXT sous OSEK

But du TP

Nous programmerons un robot LEGO™ NXT. Ce robot est en fait un « jouet » très évolué puisqu'il permet une programmation multitâche aussi bien en LabVIEW qu'en C sur exécutif OSEK. L'idée est de programmer le robot de sorte à ce qu'il suive une piste, puis de modifier le système de sorte à ce que le robot se mette à reculer lorsqu'il détecte un obstacle proche au capteur ultrason, et se remette à avancer lorsqu'il détecte un choc sur son capteur arrière. Enfin, nous ferons une étude dynamique de durée moyenne d'exécution afin d'étudier l'ordonnabilité du système, et de choisir les périodes d'exécution des tâches au plus juste par rapport à la puissance de calcul afin d'augmenter la réactivité du système.

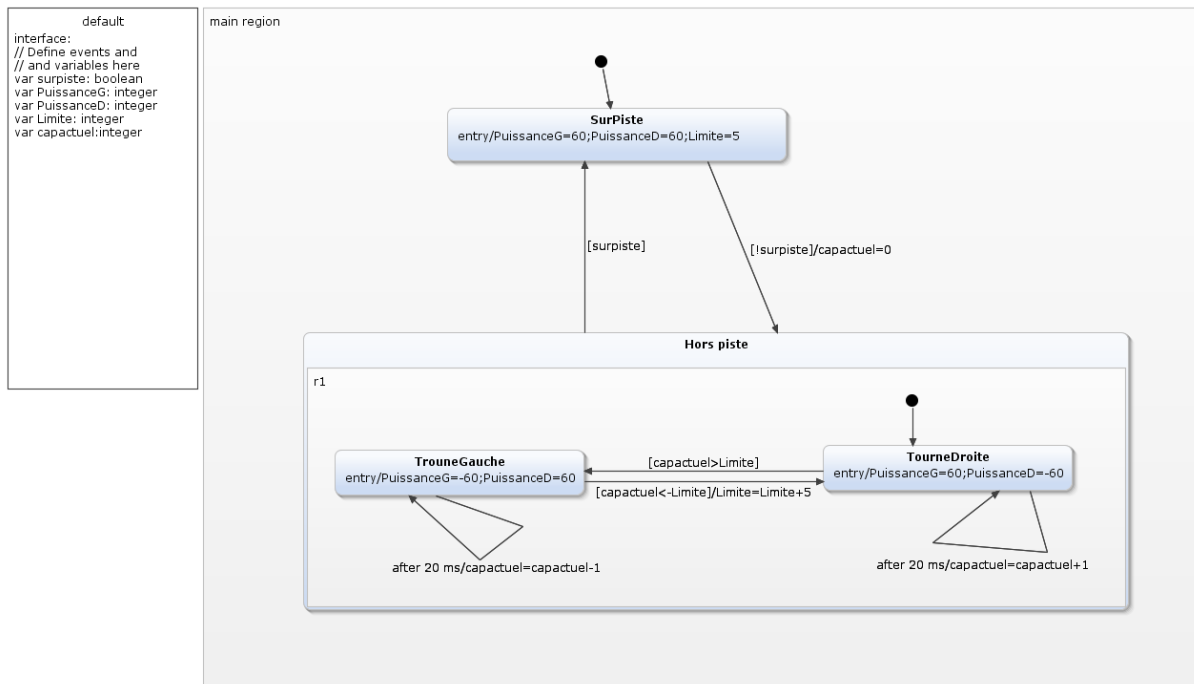
Conception



En vert, l'objectif de la 1^{ère} séance, en rose, l'application à la fin de la seconde séance.

Il y a différentes stratégies de suivi de piste, la plus simple consiste à faire des balayages de plus en plus larges à droite et à gauche. Ce comportement voulu est illustré sur le statechart ci-après. Remarquons plusieurs choses :

- **SuivrePiste** fonctionnant normalement à une période de 20 ms, nous avons choisi ce temps comme illustrant le fait que, lorsque le robot est hors piste, pendant une période, les moteurs ont tourné en sens opposé, faisant pivoter le robot sur lui-même.
- La « Limite » représente environ 100 ms pendant lesquelles le robot est allé dans un sens. A chaque fois qu'on balaie pour rien, on ajoute donc 5 à la limite (comme nous avons une période de 20 ms, cela correspondra à ajouter ~100 ms).
- Nous avons un état composite, il vaut mieux implémenter le statechart dans lequel l'état composite est « mis à plat ». Finalement, un état composite ne sert qu'à factoriser des transitions, et à exprimer une priorité.



Généralités sur l'environnement de développement employé

1. Récupérer la machine virtuelle pour virtualbox donnée. Le mot de passe pour l'utilisateur **ubuntu** est **ubuntu**.

Pour information, voici la façon dont le *toolchain* a été installé. Durée incluant les erreurs et problèmes de versions : ~8h.

Les étapes sont détaillées sur http://lejos-osek.sourceforge.net/installation_linux.htm

- a. Choix qui ont fonctionné : Ubuntu 12.04 LTS, incluant GCC 4.6.3
- b. Téléchargement des sources de GCC 4.4.2, binutils 2.20.1, newlib 1.18.0, puis compilation à l'aide du script fourni d'un compilateur croisé pour générer des exécutables au format ELF pour processeur ARM qui nous servira à générer du code pour le processeur ARM Atmel 32 bits AT91SAM7S256 du robot NXT.
- c. Le compilateur croisé se trouve dans le dossier ~/gnuarm.
- d. La bibliothèque utilisée pour programmer le robot s'appelle NXTosek et est conforme à la norme Osek. Cependant, elle existe sous la forme d'un *toppers*, c'est-à-dire une surcouche de système d'exploitation existant (en l'occurrence *LeJOS*), ce qui limite ses possibilités, et notamment la granularité d'horloge à 1 ms, que nous avons à mettre à jour nous-même en nous basant sur une interruption (voir fonction **user_1ms_isr_type2** dans la documentation *nxtosek*).
- e. Noter au passage que le compilateur de fichier OIL n'étant pas *open source*, nous avons dû utiliser un exécutable pour MS Windows, *sg.exe*, que nous lançons sous linux en utilisant l'émulateur Windows Wine. Cela est transparent et géré dans le Makefile.
- f. Afin de pouvoir transférer l'exécutable sur le processeur du NXT, nous utiliserons NeXTTool, qui se trouve dans le dossier ~/libnxt-0.3. Il a, lui aussi, été récupéré et compilé dans la machine virtuelle.
- g. L'environnement de développement est Eclipse, dans sa version CDT.

2. Lancer la machine virtuelle, se connecter (mot de passe **ubuntu**) puis lancer Eclipse

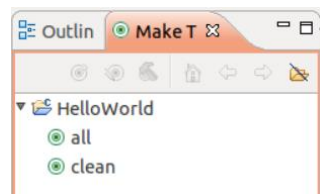


3. Il existe un projet **helloworld** qui nous servira de base. En effet, lorsqu'on fait un projet pour nxtosek, il faut avoir conscience du fait que l'application utilisateur sera compilée, ainsi que le fichier OIL, et le système d'exploitation lui-même. Les outils étant relativement non standards (par exemple sg.exe), les Makefile sont relativement complexes. Il est donc conseillé de démarrer un nouveau projet à l'aide d'un exemple, qui contient les Makefile adéquats. Nous avons donc créé dans la machine virtuelle un nouveau projet à partir de l'exemple helloworld qui se trouve dans le dossier `~/nxtosek/samples_c/helloworld`, à partir duquel nous allons créer le programme demandé.

4. Applications


- a. Shell : terminal dans la barre de menu
- b. IDE : Eclipse dans la barre de menu
- c. Navigation : Firefox dans la barre de menu
- d. Documentation nxtosek et Osek : favoris dans Firefox

5. Travailler sur un projet

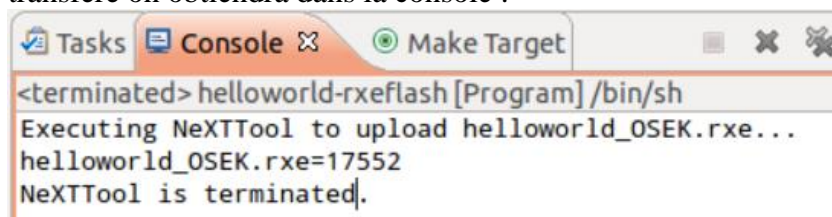


- a. Comme le projet est de type Makefile, dans la fenêtre Make Target. Dérouler le dossier HelloWorld. Double cliquer sur « all » pour compiler/générer l'exécutable, ou sur clean pour supprimer tous les fichiers créés lors de la compilation. Attention, les dépendances ne sont pas toutes exprimées dans les Makefile, par exemple, ceux-ci n'expriment pas la dépendance fichier OIL -> fichier kernel_cfg.c et kernel_id.h. Par conséquent, lorsqu'on modifie le fichier OIL, il faut faire « clean », car le Makefile ne sait pas que ces fichiers dépendent du fichier OIL. Lorsqu'on modifie les fichiers .c, on peut se contenter de faire « all », si l'on modifie le .oil, il faut faire « clean » puis « all ».

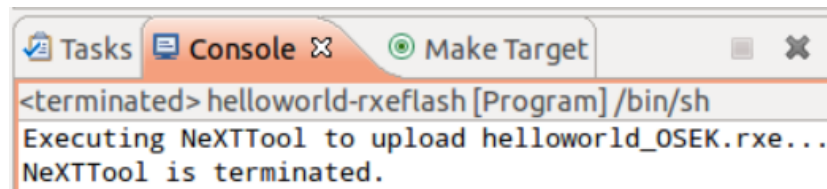
- b. Flasher l'exécutable sur le robot :

- i. Connecter un robot allumé via USB
- ii. Dans le menu Périphériques de VirtualBox, dans le sous-menu Périphériques USB, cocher « Périphérique inconnu 0694:0002 ». Cela permet à la connexion USB d'être visible dans le Ubuntu hébergé par VirtualBox.
- iii. Flasher l'exécutable à l'aide de la commande `helloworld-rxeflash` d'Eclipse .

- iv. L'utilitaire NeXTTool n'affiche pas d'erreur dans le cas où il échoue à transférer le programme. Il faut donc vérifier dans la console d'Eclipse que le retour contient bien 3 lignes, incluant une ligne « **helloworld_OSEK.rxe=...** ». Ainsi, si l'exécutable a bien été transféré on obtiendra dans la console :




Alors qu'en cas d'erreur lors du transfert on verra :



- v. Note : si le projet est relocalisé dans un autre dossier, il faudra modifier la commande externe helloworld-rxeflash.

Travail à réaliser :

1. Vérifier le fonctionnement de l'environnement de développement en compilant helloworld (Make Target all), connecter un robot NXT allumé via USB, le rendre visible dans VirtualBox, puis appliquer helloworld-rxeflash . Pour exécuter le programme sur le robot, sur la brique NXT, aller dans My Files, puis Software files, lancer helloworld_OSEK. Après affichage de l'écran OSEK, appuyer sur la flèche droite pour lancer le programme. Le bouton rectangulaire gris termine l'OS nxtosek et donc le programme par la même occasion.
2. Réaliser l'implémentation de l'étape 1 (en vert).
Erreurs fréquentes et conseils :
 - Le capteur utilisé pour suivre la piste est un capteur infrarouge et pas un capteur lumineux
 - Penser à déclarer les variables rémanentes de tâche en « static »
 - Penser à déclarer les variables partagées par des tâches en « volatile », et à les protéger par sémaphore (**resource**)
 - Il faut mettre à jour le compteur avec SignalCounter dans la fonction user_1ms_isr_type2 (voir documentation rubrique nxtOSEK Hook routines)
 - Mettre les initialisations capteur dans la fonction ecrobot_device_initialize (voir documentation rubrique nxtOSEK Hook routines)
 - Une erreur très fréquente avec Osek est d'oublier TerminateTask (ou ChainTask) à la fin d'une tâche. Vérifier que chaque tâche se termine (ou chaîne) bien.
3. Réaliser l'implémentation de l'étape 2 (en rose).

Questions bonus :

4. Mesurer les durées d'exécution moyennes des tâches. Pour cela on code chaque contenu de tâche dans une fonction, ensuite on crée une application ne contenant qu'une tâche appelant 1000 fois chaque fonction en commençant par la tâche d'affichage, puis affichant, en millisecondes, pour chaque fonction, la durée de l'exécution des 1000 appels. Il suffit alors d'interpréter le résultat en micro-secondes comme la durée moyenne d'exécution.
5. Fixer les périodes des tâches de sorte à obtenu une charge moyenne inférieure à 69%, et fixer les priorités de sorte à ce qu'elles soient inversement proportionnelles à la période. Attention, la tâche lisant l'ultrason ne pourra pas voir sa période diminuer en dessous de 10 ms.
6. Pourquoi faire ce choix de 69% ?
7. Est-ce que l'on peut considérer le programme validé temporellement, discuter de ce point.