# Compte Rendu

# TP – Système Embarqué Temps Réel

Membres du groupe :

* TRAN Van Duc
* YIN Yiliang

## Introduction

On s’attache dans ce compte rendu à décrire et à analyser le travail de notre groupe dans deux TPs de Système Embarqué Temps Réel :

* TP1 : Programmation d’un robot NXT sous OSEK
* TP2 : POSIX programming on the EV3 robot

Ces TPs nous aident à comprendre le principe d’un système embarqué multitâche en utilisant la norme OSEK, la communication et l’ordonnancement des tâches avec l’aide du langage C.

## TP1 – Programmation d’un robot NXT sous OSEK

### But du TP

Ce TP joue sur la programmation d’un robot LEGO NXT qui est équipé avec des capteurs de l’infrarouge, de l’ultrason et du choc. L’idée est de programmer le robot de sorte à ce qu’il suive les spécifications suivantes :

* Le fonctionnement principal est de suivre une piste noire avec l’aide du capteur de l’infrarouge.
* Le robot se met à reculer lorsqu’il détecte un obstacle proche au capteur ultrason.
* Il avance lorsqu’il détecte un choc sur son capteur arrière.

Enfin, nous ferons une étude dynamique de durée moyenne d’exécution afin d’étudier l’ordonnançabilité 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. La conception de ce système est décrite par le diagramme ci-dessous :

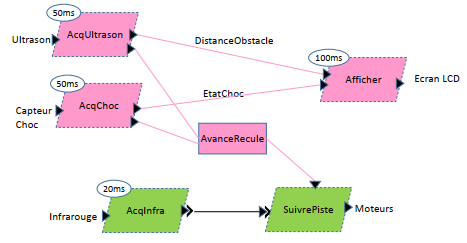


Figure 1 : Conception du Système suivi piste

Pour le réaliser, on décide à deviser le système en deux partie :

* En vert est la stratégie principale, suivi piste.
* La partie en rose sert à détecter les obstacles autour le robot, réagir sur ces informations et les afficher sur l’écran LCD.

Il faut programmer en même temps, les tâches dans la source de code en langage c et la communication entre nos tâches dans le fichier .oil (OSEK implementation language).

### Travail réalisé

Question 1 : Vérifier le fonctionnement de code.

Cette étape décrit comment compiler et lancer la source de code sur notre robot.

Question 2 : L’implémentation de l’étape 1 (partie en verte).

Ici, on constate deux tâches :

* Task\_AcqInfra
* Task\_SuivrePiste

D’abord, il faut déclarer ces deux tâches en norme OSEK. Elles partagent une même ressource sMonMDD.

TASK Task\_AcqInfra

{

AUTOSTART = FALSE;

PRIORITY = 1; /\* lowest priority \*/

ACTIVATION = 1;

SCHEDULE = FULL;

STACKSIZE = 512;

RESOURCE = sMonMDD;

};

TASK Task\_SuivrePiste

{

AUTOSTART = FALSE;

PRIORITY = 1; /\* lowest priority \*/

ACTIVATION = 1;

SCHEDULE = FULL;

STACKSIZE = 512;

RESOURCE = sMonMDD;

};

Ensuite, la ressource partagée, les deux tâches et la variable volatile pour échanger des données sont déclarées dans le fichier .c.

DeclareResource(sMonMDD);

DeclareTask(Task\_AcqInfra);

DeclareTask(Task\_SuivrePiste);

volatile int resourceinfra;

La communication entre deux tâches est synchrone de type event-data. Dans la tâche Task\_AcqInfra après d’avoir initialisé l’infrarouge, la variable val\_mdd obtient ses données puis les envoie à la tâche suivant qui est Task\_SuivrePiste en utilisant ChainTask.

La tâche Task\_SuivrePiste d’abord, associe avec Task\_AcqInfra par le code ci-dessous.

// Ici, on décrit seulement la partie initiale de Task\_SuivrePiste

TASK (Task\_SuivrePiste)

{

int val\_mdd;

GetResource(sMonMDD);

val\_mdd = resourceinfra;

ReleaseResource(sMonMDD);

ReleaseResource(avanceRecule);

TerminateTask();

}

En ce moment, ces étapes sont suffisants pour initialiser une communication event data. On considère le fonctionnement logique du système dans la suite en analysant aussi la besoin de la partie complémentaire (décrire en rose).

Question 3 : L’implémentation de l’étape 2 (partie en rose).

TASK (Task\_AcqInfra)

{

ecrobot\_device\_initialize();

int val\_mdd;

val\_mdd = ecrobot\_get\_light\_sensor(light\_sensor);

GetResource(sMonMDD);

resourceinfra = val\_mdd;

ReleaseResource(sMonMDD);

ChainTask(Task\_SuivrePiste);

}

Comme la question avant, il faut toujours énoncer nos tâches dans oil et c.

Enoncer les ressources et Task\_AcqUltrason :

RESOURCE distanceObstacle {RESOURCEPROPERTY = STANDARD;};

RESOURCE etatChoc {RESOURCEPROPERTY = STANDARD;};

RESOURCE avanceRecule {RESOURCEPROPERTY = STANDARD;};

TASK Task\_AcqUltrason

{

AUTOSTART = FALSE;

PRIORITY = 1; /\* lowest priority \*/

ACTIVATION = 1;

SCHEDULE = FULL;

STACKSIZE = 512;

RESOURCE = distanceObstacle;

RESOURCE = avanceRecule;

};

ALARM cyclic\_alarmAcqUltrason

{

COUNTER = SysTimerCnt;

ACTION = ACTIVATETASK

{

TASK = Task\_AcqUltrason;

};

AUTOSTART = TRUE

{

ALARMTIME = 1;

CYCLETIME = 50;

APPMODE = appmode1;

};

};

Task\_AcqChoc

TASK Task\_AcqChoc

{

AUTOSTART = FALSE;

PRIORITY = 1; /\* lowest priority \*/

ACTIVATION = 1;

SCHEDULE = FULL;

STACKSIZE = 512;

RESOURCE = etatChoc;

RESOURCE = avanceRecule;

};

ALARM cyclic\_alarmAcqChoc

{

COUNTER = SysTimerCnt;

ACTION = ACTIVATETASK

{

TASK = Task\_AcqChoc;

};

AUTOSTART = TRUE

{

ALARMTIME = 1;

CYCLETIME = 50;

APPMODE = appmode1;

};

};

Task\_Afficher

TASK Task\_Afficher

{

AUTOSTART = FALSE;

PRIORITY = 1; /\* lowest priority \*/

ACTIVATION = 1;

SCHEDULE = FULL;

STACKSIZE = 512;

RESOURCE = distanceObstacle;

RESOURCE = etatChoc;

};

ALARM cyclic\_alarmAfficher

{

COUNTER = SysTimerCnt;

ACTION = ACTIVATETASK

{

TASK = Task\_Afficher;

};

AUTOSTART = TRUE

{

ALARMTIME = 1;

CYCLETIME = 100;

APPMODE = appmode1;

};

};

Les mêmes étapes sont appliqués pour les codes c.

Enoncer les ressources et Task\_AcqUltrason :

/\* task ultrason pour reculer\*/

DeclareResource(distanceObstacle);

DeclareResource(avanceRecule);

DeclareResource(etatChoc);

DeclareTask(Task\_AcqInfra);

DeclareTask(Task\_AcqUltrason);

DeclareTask(Task\_AcqChoc);

DeclareTask(Task\_Afficher);

TASK (Task\_AcqUltrason)

{

ecrobot\_device\_initialize();

int val\_ultrason;

val\_ultrason = ecrobot\_get\_sonar\_sensor(ultrason\_sensor);

GetResource(distanceObstacle);

GetResource(avanceRecule);

resourcedistance = val\_ultrason;

if(val\_ultrason < seuil\_distance)

{

resourceAR = -1;

} else {

resourceAR = 1;

}

ReleaseResource(distanceObstacle);

ReleaseResource(avanceRecule);

TerminateTask();

}

Task\_AcqChoc :

/\* task choc pour avancer\*/

TASK (Task\_AcqChoc)

{

ecrobot\_device\_initialize();

int val\_choc;

val\_choc = ecrobot\_get\_touch\_sensor(choc\_sensor) ;

GetResource(etatChoc);

GetResource(avanceRecule);

resourcechoc = val\_choc;

if(val\_choc==1)

{

resourceAR = 1;

}

ReleaseResource(etatChoc);

ReleaseResource(avanceRecule);

TerminateTask();

}

Task\_Afficher :

TASK(Task\_Afficher)

{

int val\_do;

GetResource(distanceObstacle);

val\_do = resourcedistance;

int val\_choc;

GetResource(etatChoc);

val\_choc = resourcechoc;

display\_goto\_xy(0,0);

display\_string("DO:");

display\_int(val\_do,3);

display\_goto\_xy(0,3);

display\_string("EC:");

display\_int(val\_choc,1);

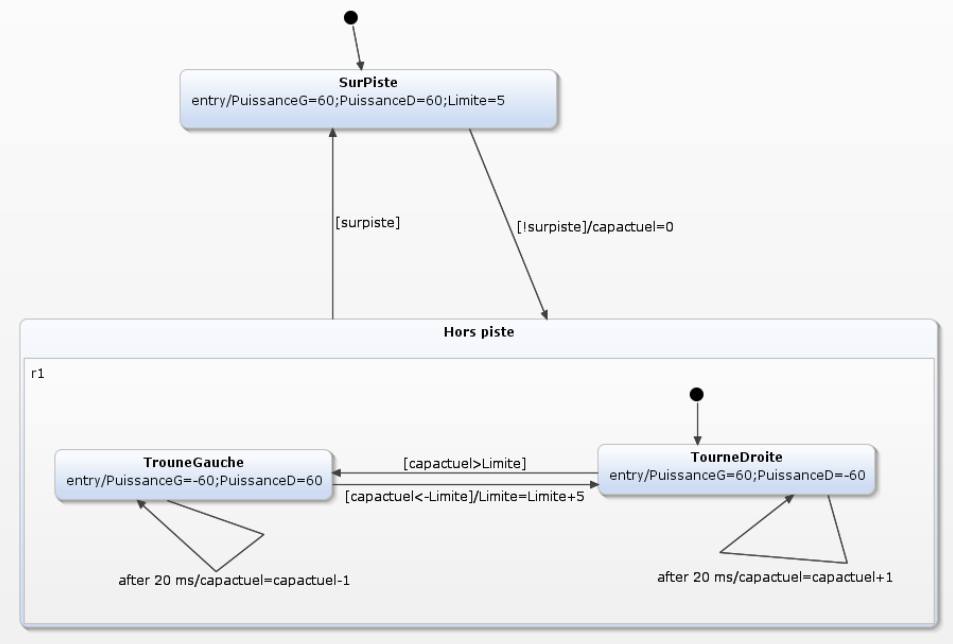
ReleaseResource(distanceObstacle);

ReleaseResource(etatChoc);

TerminateTask();

}

Pour compléter la logique de la fonction, on considère le statechart



En exploitant le statechart, on obtient le code ci-dessous. Etat a correspond avec l’état SurPiste. Si le robot est « hors piste », il change à l’état b, TourneDroite (état original hors piste). Si le capteur arrive à sa limite mais il ne trouve pas la piste, il tourne à gauche (état c).

TASK (Task\_SuivrePiste)

{

static int ar = 1;

GetResource(avanceRecule);

ar = resourceAR;

int val\_mdd;

GetResource(sMonMDD);

val\_mdd = resourceinfra;

static int etat = etat\_a;

static int capactuel;

static int limite = 5;

switch(etat) {

case etat\_a :/\*etat original surpiste\*/

if(val\_mdd > seuil\_infra) { /\*surpiste\*/

nxt\_motor\_set\_speed(roue\_gauche,60\*ar,1);

nxt\_motor\_set\_speed(roue\_droite,60\*ar,1);

}

else { /\*hors piste\*/

etat = etat\_b; /\*tourner droite\*/

nxt\_motor\_set\_speed(roue\_gauche,60\*ar,1);

nxt\_motor\_set\_speed(roue\_droite,-60\*ar,1);

capactuel = 0;

}

break;

Etat b :

case etat\_b : /\*etat original hors piste\*/

if(val\_mdd > seuil\_infra){ /\*surpiste\*/

nxt\_motor\_set\_speed(roue\_gauche,60\*ar,1);

nxt\_motor\_set\_speed(roue\_droite,60\*ar,1);

limite = 5;

etat= etat\_a;

} else if (capactuel > limite) {

/\*hors piste\*/

/\*tourner gauche\*/

nxt\_motor\_set\_speed(roue\_gauche,-60\*ar,1);

nxt\_motor\_set\_speed(roue\_droite,60\*ar,1);

etat = etat\_c;

} else {

capactuel = capactuel+1;

}

break;

Etat c et la fin de Task\_SuivrePiste :

case etat\_c :

if(val\_mdd > seuil\_infra){ /\*surpiste\*/

nxt\_motor\_set\_speed(roue\_gauche,60\*ar,1);

nxt\_motor\_set\_speed(roue\_droite,60\*ar,1);

limite = 5;

etat = etat\_a;

} else if (capactuel < -limite) {

/\*hors piste\*/

/\*tourner droite\*/

nxt\_motor\_set\_speed(roue\_gauche,60\*ar,1);

nxt\_motor\_set\_speed(roue\_droite,-60\*ar,1);

limite = limite+5;

etat = etat\_b;

} else {

capactuel = capactuel-1;

}

}

ReleaseResource(sMonMDD);

ReleaseResource(avanceRecule);

TerminateTask();

## TP2 – POSIX programming on the EV3 robot

## Conclusion