

UBO

lab-sticc.univ-brest.fr/~babau/

Programmation NXC
Application à la commande d'un robot
NXT Lego®

Jean-Philippe Babau

Département Informatique, UFR Sciences, UBO
Laboratoire Lab-STICC

jean-philippe.babau@univ-brest.fr

UBO

jean-philippe.babau@univ-brest.fr

Plan

- “ Caractéristiques du robot NXT de Lego®
- “ Programmation du robot NXT
 - . Gestion des entrées/sorties
- “ Not eXactly C : pas exactement du C
 - . Langage de programmation proche du langage C
 - . Programmation des briques NXT de Lego ®
- “ Programmation multi-tâches en NXC
 - . Temps, tâches et synchronisation

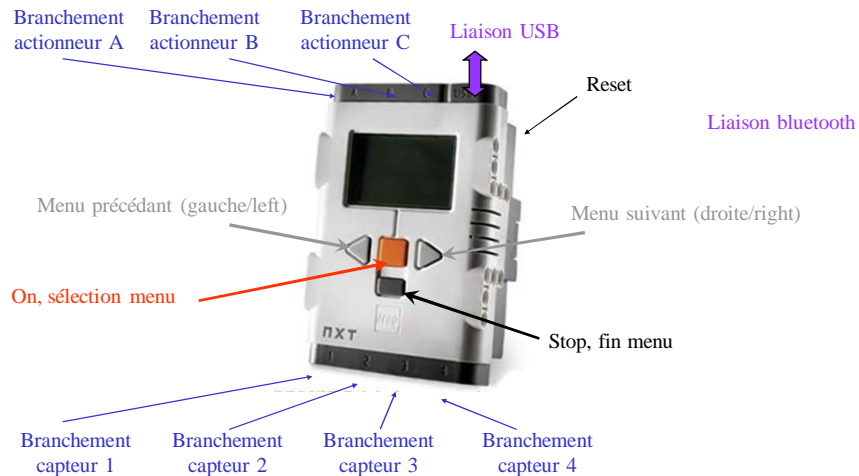
Objectifs du cours

- “ Commande d'un robot NXT (Lego Mindstorms®)
 - . quatre types de capteur (contact, luminosité, Ultra Son, micro)
 - . trois actionneurs (3 moteurs)
 - “ Capteurs de suivi (tours, puissance)
 - . un générateur de son
 - . quatre boutons
 - . un écran LCD
 - . une connexion bluetooth
 - . une connexion USB
- “ Structure d'un programme NXC
 - . main, données, algorithmes, fonctions
- “ Structure d'un programme multi-tâches NXC
 - . Temps, tâches, synchronisation

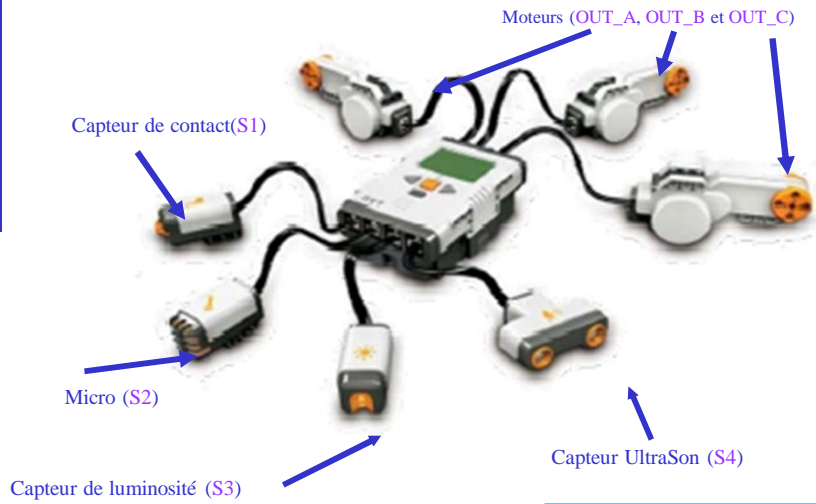
Plan

- “ Caractéristiques du robot NXT de Lego®
- “ Programmation du robot NXT
 - . Gestion des entrées/sorties
- “ Not eXactly C : pas exactement du C
 - . Langage de programmation proche du langage C
 - . Programmation des briques NXT de Lego ®
- “ Programmation multi-tâches en NXC
 - . Temps, tâches et synchronisation

Le robot NXT



Le robot et ses périphériques



jean-philippe.babau@univ-brest.fr

Block Diagram

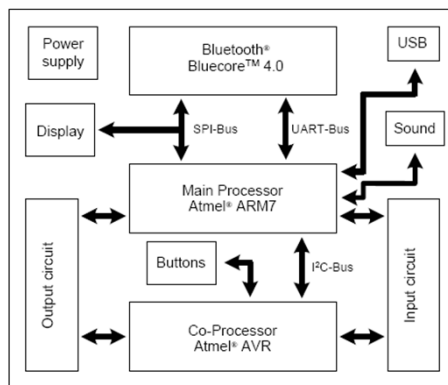
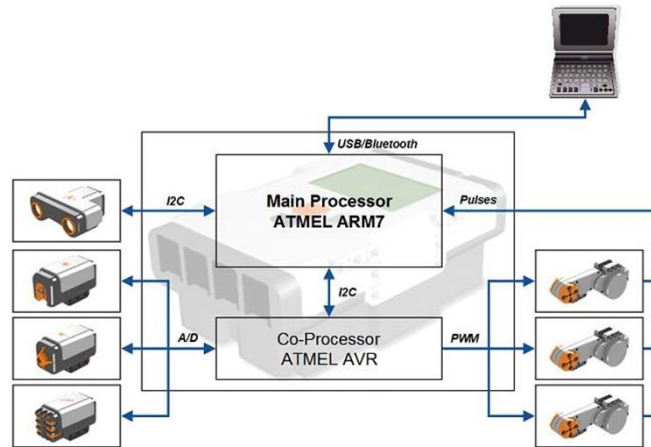


Figure 1: Hardware block diagram of the NXT brick

- Atmel® 32-bit ARM7® processor
 - ó AT91SAM7S256
 - ó 256 Ko FLASH (1000 pages de 256 octets), 64 Ko RAM, 48 MHz
- Atmel® 8-bit AVR processor
 - ó ATmega48
 - ó 4 Ko FLASH, 512 Octets RAM, 8 MHz
- CSR BlueCore™ 4 v2.0
 - ó 47 Ko RAM, 8 MBit FLASH, 26 MHz
 - ó Supporting the Serial Port Profile (SPP)
 - ó full speed port (12 Mbit/s)

jean-philippe.babau@univ-brest.fr

Schéma des connexions



jean-philippe.babau@univ-brest.fr

Architecture matérielle et logicielle

Logiciels

- . Bootcode
 - ~ Initialisation de la carte
 - ~ Gestion des entrées, sorties
 - ~ Communication USB
- . Firmware
 - ~ API du lego
 - ~ Services d'accès aux entrées et aux sorties
 - ~ Exécutif multitâches
 - ~ Par défaut : code interprété

C:\Program Files\LEGO Software\LEGO MINDSTORMS NXT\engine\Firmware\ LEGO MINDSTORMS NXT\Firmware v1.05.rfw

~ Autres firmwares : leJOS(NXJ), robotc(C) (lejos.sourceforge.net, www.robotc.net)

Code applicatif

firmware

bootcode

CPU

Mémoires

- . ROM : mémoire persistante, en lecture seule
- . Flash : mémoire persistante, en lecture / écriture
 - . Fichiers (programmes, log, traces, images et sons)
- . RAM : mémoire volatile (persistante si alimentée)
 - . Exécution des programmes

RAM

Flash

ROM

jean-philippe.babau@insa-lyon.fr

Programmation

- “ Développement croisé
 - . Machine hôte (PC, Mac) : développement et mise au point
 - . Machine cible (NXT) : exécution d'un programme
 - . Liaison USB : téléchargement, suivi *et communications*

- “ Environnement de développement (*Bricx Command Center* ou *BricxCC*)
 - . Éditeur de langage NXC
 - . Compilation (*Compile*) et téléchargement (*Download*)
 - . Diagnostic de la carte
 - . Gestion distante des fichiers et des programmes (*NeXT Explorer*)
 - . Suivi des variables (*Watch*)
 - . Contrôle à distance (*Tools/Diagnostics*)
 - “ Lejos : librairie de services JAVA de commandes distants

 - . Sous Vista : Ordinateur -> C:\Programmes\BricxCC\BricxCC.exe
 - . Plus d'informations sur <http://bricxcc.sourceforge.net/>

- “ Simulation
 - . Simulation de l'exécution sur machine hôte avec Lejos

Quelques problèmes í

- “ Le robot n'est pas visible par le logiciel
 - . Allumer le robot !
 - . *Tools / Find Brick*

- “ Plus de piles
 - . Penser à re-télécharger le firmware (?)

- “ Reset (bouton à l'arrière du robot, niveau port USB)
 - . Rester appuyé au moins 2 secondes
 - . Penser à re-télécharger le firmware (?)

- “ Téléphones, PDA, ð
 - . Interférences avec la liaison bluetooth
 - . Interférence IR avec le capteur de luminosité

Plan

- “ Caractéristiques du robot NXT de Lego®
- “ Programmation du robot NXT
 - . Gestion des entrées/sorties
- “ Not eXactly C : pas exactement du C
 - . Langage de programmation proche du langage C
 - . Programmation des briques NXT de Lego ®
- “ Programmation multi-tâches en NXC
 - . Temps, tâches et synchronisation

Les capteurs : configuration

- “ 4 ports de connexion (numérotés 0,1, 2 et 3) appelés
S1, S2, S3 et S4

É Un type

SENSOR_TYPE_NONE	capteur générique passif
SENSOR_TYPE_TOUCH	capteur de contact
SENSOR_TYPE_LIGHT	capteur de luminosité
SENSOR_TYPE_SOUND_DB	capteur de son
SENSOR_TYPE_LOWSPEED	capteur I2C (capteur Ultra Son)
SENSOR_TYPE_ROTATION	capteur de rotation

É Un mode

SENSOR_MODE_RAW	intervalle entre 0 et 1023
SENSOR_MODE_BOOL	booléen (0 ou 1)
SENSOR_MODE_EDGE	nombre de transitions booléennes
SENSOR_MODE_PULSE	nombre de périodes booléennes
SENSOR_MODE_PERCENT	valeur entre 0 et 100

- “ Une configuration : un type + un mode
configuration

SENSOR_TOUCH
SENSOR_LIGHT
SENSOR_PULSE

Type

SENSOR_TYPE_TOUCH
SENSOR_TYPE_LIGHT
SENSOR_TYPE_TOUCH

Mode

SENSOR_MODE_BOOL
SENSOR_MODE_PERCENT
SENSOR_MODE_PULSE

Les capteurs : 3 initialisations possibles

“ Initialiser via le type et le mode

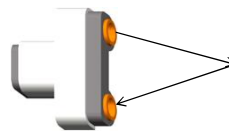
```
SetSensorType(portNumber, type);           SetSensorType(S1, SENSOR_TYPE_TOUCH);
SetSensorMode(portNumber, mode);         SetSensorMode(S1, SENSOR_MODE_BOOL);
```

“ Ou initialiser via la configuration

```
SetSensor(portNumber, configuration);     SetSensor(S1, SENSOR_TOUCH);
```

“ Ou initialiser via une procédure spécifique

```
SetSensorTouch(portNumber);              SetSensorTouch(S1);
// capteur de contact, 1 appuyé; 0 : relâché
SetSensorLight(portNumber);
// capteur de luminosité, échelonné de 0 à 100
SetSensorSound(portNumber);
// capteur de volume de son, échelonné de 0 à 100
SetSensorLowspeed(portNumber);
// capteur Ultra Son, étalonné en cm (0-255 : proc. 8 bits)
```



Les capteurs : les actions

“ Récupérer une information d'un capteur

```
int Sensor(portNumber);                 // renvoie la valeur du capteur n
int SensorUS(portNumber);               // renvoie la valeur du capteur I2C (capteur Ultra Son)
```

“ Exemple

```
int appuye;
SetSensorTouch(S1);
appuye = Sensor(S1) // capteur appuyé : 1, relâché 0
```

“ Remettre à 0 un capteur cumulatif

```
ClearSensor(portNumber);                ClearSensor(S1);
```


UBO

Les capteurs spécifiques

~ Capteur de direction (compas)

Attention : le capteur doit rester horizontal

```
SetSensorLowspeed(portNumber); // initialise le capteur
int SensorHTCompass(portNumber); // renvoie la direction en degrés entre 0 et 359
```



~ Capteur infra-rouge(compas)

```
SetSensorLowspeed(portNumber); // initialise le capteur
int ReadSensorHTIRSeeker2AC(const byte port, byte &dir, byte &s1, byte &s3,
byte &s5, byte &s7, byte &s9 ); // renvoie la direction et le signal lue dans les
directions 1, 3, 5 et 7
```



~ Accéléromètre

```
SetSensorLowspeed(portNumber); // initialise le capteur
int ReadSensorHTAccel(portNumber,int &x, int &y, int &z); // renvoie l'accélération
dans les 3 directions
```



~ Gyroscope

```
SetSensorHTGyro(portNumber); // initialise le capteur
int SensorHTGyro(portNumber); // renvoie l'offset
int SensorHTGyro(portNumber, offset); // renvoie la vitesse angulaire en degré / s
```



jean-philippe.babau@univ-brest.fr

UBO

Les boutons

~ Les boutons (0 à 3)

```
BTNEXIT, BTNRIGHT, BTNLEFT et BTNCENTER
BTN1, BTN2, BTN3 et BTN4
```

~ Les actions

```
int ButtonCount(buttonNumber, reset);
// renvoie le nombre de fois où le bouton buttonNumber a été appuyé, remise a 0 si reset est égal à true
```

```
int ButtonPressed(buttonNumber, reset);
// renvoie si le bouton buttonNumber est appuyé, remise a 0 du nombre d'appui si reset est égal à true
```

jean-philippe.babau@univ-brest.fr

UBO

Les actionneurs

- " 3 actionneurs, 4 combinaisons d'actionneurs (numérotés 0,1 et 2; puis 3,4,5 et 6)
OUT_A, OUT_B, OUT_C
OUT_AB, OUT_AC, OUT_BC, OUT_ABC
- " Commande en puissance
 - . pourcentage % [-100,100]
 - OnFwd(outputs, power);** OnFwd(OUT_A, 75); // OUT_A en avant à 75%
 - OnFwd(OUT_A, -75); // OUT_A en arrière à 75%
 - OnRev(outputs, power);** OnRev(OUT_AC, 75); // OUT_A et OUT_C en arrière à 75%
- " Arrêt
 - Float(outputs);** // arret avec freinage
 - Off(outputs);** // arret sans freinage

jean-philippe.babau@univ-brest.fr

UBO

Les actionneurs : les actions spécifiques

- " Commande en position
 - RotateMotor(outputs, power,angle);**
 - RotateMotor(OUT_C, 75, -180); // OUT_C fait un demi tour arrière
- " Commande PID en position
 - RotateMotorPID(outputs, power,angle,P,I,D);**
 - P : gain proportionnel, I : effet mémoire, D : évolution (dérivée)
- " Commande en vitesse
 - OnFwdReg(outputs, power, OUT_REGMODE_SPEED) ;**
 - // la vitesse est maintenue constante (accélération si nécessaire)
- " Commande synchronisée
 - OnFwdReg(outputs, power, OUT_REGMODE_SYNC) ;**
 - // la vitesse de tous les moteurs désignés par outputs est identique

jean-philippe.babau@univ-brest.fr

UBO

Tests PID

É Primitive : **RotateMotorPID**(OUT_A , 100 , 360, P, I, D); // OUT_A fait un tour à vitesse maximale

RotateMotorPID(OUT_A , 100 , 360, 50, 0 , 0);

RotateMotorPID(OUT_A , 100 , 360, 200, 0 , 0);

RotateMotorPID(OUT_A , 100 , 360, 1000, 0 , 0);

RotateMotorPID(OUT_A , 100 , 360, 0, X , X);

RotateMotorPID(OUT_A , 100 , 360, 40, 40 , 0);

RotateMotorPID(OUT_A , 100 , 360, 40, 40 , 90);

RotateMotorPID(OUT_A , 100 , 360, 40, 40 , 200);

jean-philippe.babau@univ-brest.fr

UBO

Suivi des moteurs

~ Suivi de la commande

MotorActualSpeed(output); // renvoie la puissance envoyée sur le moteur output

~ Suivi des rotations effectuées

- . Angles
- . **Attention** : compte remis à 0 à chaque nouvelle commande

MotorTachoCount(output); // renvoie le nombre de degrés effectués sur le moteur output

ResetTachoCount(output); // remise à 0 du nombre de degrés sur le moteur output

jean-philippe.babau@univ-brest.fr

UBO

Le son

- ~ 1 générateur de son
 - . Un son à la fois, attendre la fin du précédent pour jouer le suivant
- ~ Jouer une mélodie
 - PlayFileEx("soundFile.rso ", volume, repeat);**
 - // jeu de file.rso avec un volume de 0 à 4, repeat à true pour jouer en boucle
 - . Création d'un fichier .rso avec BrickPiano de BricxCC
 - . Création d'un fichier rso avec l'outil wav2rso (accessible sur le site web de lego)
- ~ Jouer une fréquence
 - PlayToneEx(frequency, duration, volume, bloop);**
 - PlayTone(frequency, duration);**
 - // duration en ms, bloop à true pour un son en boucle
 - PlayTone(440, 1000); // joue do durant 1 seconde
- ~ Arrêter
 - StopSound();**

jean-philippe.babau@univ-brest.fr

UBO

L'affichage via l'écran LCD

- ~ Caractéristiques
 - . 100 pixels en largeur x 64 pixels en hauteur
 - . 1 caractère : 8 pixels en hauteur, 6 pixels en largeur
 - . 8 lignes (LCD_LINE1, ð , LCD_LINES)
 - ~ Une ligne : 16 caractères
- ~ Affichage
 - ó X colonne, Y : ligne
 - TextOut(X, Y, string, clear);**
// affiche string à partir de la position X,Y, efface l'écran si clear est égal à true
 - NumOut(X, Y, value, clear);**
// affiche value à partir de la position X,Y, efface l'écran si clear est égal à true
 - ClearScreen(); // efface l'écran**
 - GraphicOut(X, Y, "GraphicFile.ric");**
 - PointOut(X, Y);LineOut(X1, Y1, X2, Y2); CircleOut(X, Y, R); RectOut(X, Y, largeur, hauteur);**

jean-philippe.babau@univ-brest.fr

L'affichage via l'écran LCD

~ Exemple

```
#define largChar 6

ClearScreen(); // efface l'écran
TextOut(0, LCD_LINE3, "test LCD : ", false); // affiche test LCD :
NumOut(6*11, LCD_LINE3, 11, false); // affiche test LCD : 11
NumOut(largChar*11, LCD_LINE3, 2, false); // affiche test LCD : 21 !!!
NumOut(6*11, LCD_LINE3, 12, false); // affiche test LCD : 12
NumOut(6*11, LCD_LINE3, 3, true); // affiche 3
TextOut(0, LCD_LINE3, "test LCD : ", false); // affiche test LCD : 3 !!!
NumOut(6*11, LCD_LINE3, 14, false); // affiche test LCD : 14
TextOut(0, LCD_LINE3, "test LCD : ", false); // affiche test LCD :
NumOut(6*11, LCD_LINE3, 2, false); // affiche test LCD : 2
TextOut(6*11, LCD_LINE3, " ", false); // affiche test LCD :
```

Les fichiers

~ Caractéristiques

- . Nom du fichier : 19 caractères avec extension (20 caractères en mémoire, $\pm 0q$)
- . Au 4 fichiers ouverts en même temps

~ Création et connexion

```
CodeRetour CreateFile(string fileName, short size, byte handle);
// crée le fichier filename avec la taille size et renvoie handle
CodeRetour OpenFileAppend(string fileName, short size, byte handle);
// ouvre le fichier existant filename; renvoie la taille size et handle
CodeRetour CloseFile(byte handle);
// ferme le fichier caractérisé par handle
CodeRetour DeleteFile(string fileName);
// supprime le fichier caractérisé par filename
```

~ Récupération des fichiers sur la machine de développement

- . NeXT Explorer du logiciel BricxCC
- . Si bug lors du remplissage du fichier : fichier non récupérable

UBO

Les fichiers : opérations

```
WriteBytes(handle, array, nbOctets);  
// écrit nbOctets octets du tableau array dans le fichier caractérisé par handle  
  
ReadBytes(handle, nbOctets, buffer);  
// lit nbOctets octets dans le fichier caractérisé par handle  
// place les octets lus dans le tableau buffer et renvoie dans nbOctets le nombre d'octets lus  
  
Write(handle, value);  
WriteLn(handle, value);  
// écrit value dans le fichier caractérisé par handle  
// écrit value + CR+LF dans le fichier caractérisé par handle  
  
WriteString(handle, chaine, nombreDeCaracteresEcrits);  
WriteLnString(handle, chaine, nombreDeCaracteresEcrits);  
// écrit chaine dans le fichier caractérisé par handle  
// écrit chaine + CR+LF dans le fichier caractérisé par handle  
// nombreDeCaracteresEcrits est un retour (passage par référence)  
  
Read(handle, value);  
ReadLn(handle, value);  
// lit value dans le fichier caractérisé par handle  
// lit value dans le fichier caractérisé par handle, puis lit CR et LF
```

jean-philippe.babau@univ-brest.fr

UBO

Communications Bluetooth

```
“ Connexion  
. Communication : 1 Master / 3 Slave maximum  
. Débit : 57 600 octets /seconde  
. Connexion physique via les briques  
  “ Identifiant de connexion : Master : 0, Slave : 1, 2 ou 3  
. Vérification de la connexion  
  
BluetoothStatus(connection); // renvoie NO_ERR si la connexion est ok  
  
“ Communication  
. Un message n'est considéré que si un programme s'exécute  
. 20 files circulaires (10 en entrée, 10 en sortie) par brique, 5 messages par file  
. Message de 58 octets au maximum (59 octets par message)  
  
“ Emission de chaînes de caractères  
. Master  
SendRemoteString(connection, mbxNumber, message);  
// envoi message, de type string, dans la file mbxNumber de connexion  
  
. Slave  
SendResponseString(mbxNumber, message);  
// envoi message, de type string, dans la file mbxNumber du master
```

jean-philippe.babau@univ-brest.fr

UBO

Communication

“ Emission de nombres (**autre que 0**)

. Master

`SendRemoteNumber(connection, mbxNumber, entier);`

// envoie entier, de type int, dans la file mbxNumber de connection

. Slave

`SendResponseNumber(mbxNumber, entier);`

// envoie entier, de type int, dans la file mbxNumber du master

” Réception de chaînes de caractères ou de nombre

ó **Lecture non bloquante**, renvoie 0 si pas de message

. Master et Slave

`ReceiveRemoteString(mbxNumber, clear, message);`

// place dans message, de type string, le plus ancien message de la file mbxNumber; l'efface de la file si clear est égal à true

. Master et Slave

`ReceiveRemoteNumber(mbxNumber, clear, entier);`

// place dans entier, de type int, la plus ancienne valeur de la file mbxNumber; l'efface de la file si clear est égal à true

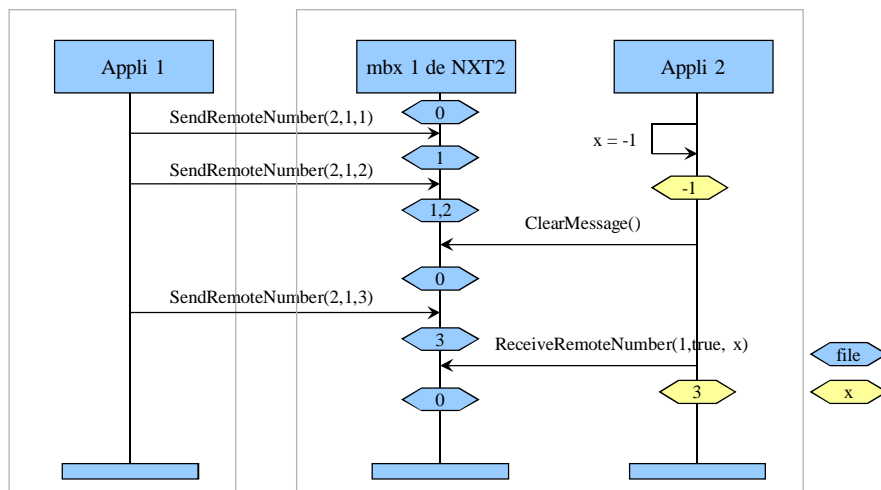
jean-philippe.babau@univ-brest.fr

UBO

Exemple de communications

Master « télécommande NXT1 »

Slave « Robot NXT2 », connection 2



jean-philippe.babau@univ-brest.fr

UBO

Communication

- “ Envoi et réception de octets
`RemoteBluetoothWrite(connection, mbxNumber, buffer);`
// envoi entier, de type int, dans la file mbxNumber de connection

`ReceiveBluetoothWrite(connection, mbxNumber, buffer);`
// envoi entier, de type int, dans la file mbxNumber de connection

- “ Envoi de commandes de haut niveau
`RemoteStartProgram(connection, fileName.rxc6);`
`RemoteSopProgram(connection);`

`RemoteSetInputMode(...); RemoteSetOutputState(...); ...`

- “ Performances
 - Placer une attente (`Wait(10);`) entre deux appels à `SendXX(f)`; ou `BluetoothStatus(f)`;
 - Placer une attente (`Wait(30);`) entre deux appels à `ReceiveXX(f)`; entre `SendXX();` et `ReceiveXX();`;
 - Débit applicatif
 - 1 entier : environ 100 ms (émission, switch réception / émission, f)

jean-philippe.babau@univ-brest.fr

UBO

Autres fonctions

- “ Générateur aléatoire
`Random();` // renvoie une valeur aléatoire sur 16 bits
`Random(n);` // renvoie une valeur aléatoire sur 16 bits entre 0 et n (non compris)

`n = Random(10);` // renvoie une valeur entre 0 et 9

- “ Niveau de batterie
`BatteryLevel();` // renvoie le niveau de batterie en millivolts

`x = BatteryLevel();`

- “ Gestion du temps
`Wait(nbTicks);` // attente de nbTicks ms

- “ Arrêt de l'application
`StopAllTasks();` // arrêt du programme
`Stop(condition);` // arrêt du programme si condition est égal a true

jean-philippe.babau@univ-brest.fr

Programme de commandes

- ~ Une tâche principale
 - . task main
 - . Séquence de commandes

```
#include "NXCDefs.h"
/* pas obligatoire mais plus pratique : définitions de noms, de macros */
/* peut être automatiquement inclus, via l'environnement de développement */

task main ()
{
    /* séquence de commandes */
}
```

Plan

- ~ Caractéristiques du robot NXT de Lego®
- ~ Programmation du robot NXT
 - . Gestion des entrées/sorties
- ~ Not eXactly C : pas exactement du C
 - . Langage de programmation proche du langage C
 - . Programmation des briques NXT de Lego ®
- ~ Programmation multi-tâches en NXC
 - . Temps, tâches et synchronisation

Programme NXC

- “ Déclarations de données
 - . Initialisation des données
 - . Définition de constantes

- “ Tâche : séquences d'instructions
 - . Commandes robot
 - . Opérations sur des données
 - . Appels de fonctions
 - . Structures algorithmiques

Introduction à NXC

- “ Not eXactly C : pas exactement du C

- “ Langage de programmation proche du langage C
 - . Syntaxe C-like

- “ Programmation des briques NXT de Lego ®
 - . Gestion des entrées / sorties

- “ Programmation multi-tâches

Éléments syntaxiques

- ~ Différence entre les minuscules et les majuscules
 - ó max et Max sont différents
 - ~ Liste de mots clés réservés
 - . Cf. suite
 - ~ Identificateur
 - . suite de lettres, chiffres, _ ne commençant pas par un chiffre
- | | | | | | |
|-------|------|-----|----|---|--------|
| Ne_le | x1 | _12 | IF | : | ok |
| 2_A | 2000 | 3A | if | : | non ok |
- ~ Commentaires
 - ó /* commentaire a la C */
 - ó // commentaire a la C++

Liste des mots clés

<u>__RETURN__</u>	<u>__RETVAL__</u>	<u>__STRRETVAL__</u>				
<u>__TMPWORD__</u>	<u>__TMPLONG__</u>	<u>__TMPBYTE__</u>				
abs	asm	bool	break	byte	case	char
const	continue	default	do	else	false	for
goto	if	inline	int	long	mutex	priority
repeat	return	safecall	short	sign	start	stop
sub	switch	task	true	typedef	unsigned	until
void	while					

Les données

- “ Tout information manipulée par le programme
 - . Constantes, variables
- “ Caractérisé par
 - . un identificateur : nom pertinent
 - . Une initialisation
 - . **un commentaire**
- “ Types entier
 - . entier sur 8 bits (1 octet) : **bool, byte, char, unsigned char**
 - . entier sur 16 bits (2 octets) : **short, int** ([-32768,32767]), **unsigned int**
 - . entier sur 32 bits (4 octets) : **long, unsigned long**
- “ Chaîne de caractère
 - . Tableau de **byte** : **string**

Les données

- “ Valeurs
 - . décimal : 16 - 33
 - . hexadécimal : 0x10
 - . booléen : false(0), true(1) ; différent de 0 est vrai
- “ Déclaration de données


```
int nombreTour; // nombre de tours effectués
int missionTerminee = 0;
int a, b ;
```
- “ Déclaration de *constantes* (macro substitution)


```
#define Annee 2007 // année courante à modifier selon la date
```
- “ Déclaration de tableaux


```
int lesPositions[10]; // ensemble des positions à atteindre
Déclaration de lesPositions[0], i , lesPositions[9]
```

UBO

Les tableaux de données

~ Déclaration de tableaux

Déclaration de lesPositions[0], f , lesPositions[1]

```
int lesPositions[] = {43, 56 } // ensemble des 2 positions à atteindre
```

```
int lesPositions[2]; // ensemble des 2 positions à atteindre
```

```
ArrayInit(lesPositions, 0, 10); // initialisation du tableau
```

```
lesPositions[0] = 43;
```

```
lesPositions[1] = 56;
```

~ **Attention aux indices**

. ErrorFile à l'exécution si non respect des indices !!

jean-philippe.babau@univ-brest.fr

UBO

Les tableaux à n dimensions

~ Initialisation via ArrayInit

~ Accès au tableau via un seul indice

~ Pas d'affectation

```
int historique1[];
```

```
int historique2[];
```

```
int coordonnees[] [];
```

```
ArrayInit (historique1,0, 10); // initialisation à 0
```

```
ArrayInit (historique2,0, 10); // initialisation à 0
```

```
ArrayInit (coordonnees, historique1, 2); // initialisation à 0
```

// pour implémenter *coordonnees[1][4] = 34*; faire

```
historique1[4] = 34;
```

```
coordonnees[1] = historique1;
```

// pour implémenter *coordonnees[0] = coordonnees[1]*; faire

```
historique2 = coordonnees [1];
```

```
coordonnees [0] = historique2;
```

jean-philippe.babau@univ-brest.fr

Les structures de données

“ Déclarations, initialisation et utilisation

```

struct robot
{
    string name;           // nom du robot
    int puissance;       // vitesse du robot
};

robot monRobot ;       // le robot

monRobot.name = " IUTA ";
monRobot.puissance = 0; // le robot est arrêté au départ

OnFwd(OUT_A, 75);       // le robot avance
monRobot.puissance = 75 ;
// affichage vitesse du robot
TextOut(0, LCD_LINE3, @Puissance demandee : " , true);
NumOut(64, LCD_LINE3, monRobot.puissance , false);

```

“ Affectation possible si même type

Les opérateurs

“ Liste d'opérateurs

+	-	*	/ (division entière)	% (modulo)
++ (incréméntation)	-- (décréméntation)			
= (affectation)				
== (comparaison)	!=	<	>	<=
! (négation logique)	&& (et logique)		(ou logique)	
<<, >> (décalages)	& (et bit à bit)	(ou bit à bit)	^ (xor bit à bit)	~ (not bit à bit)

“ exemples

5 / 4	5 % 4	32767 + 1
5 == 4	7 <= 19	
!5	5 && 2	
5 & 2	3 0	
int a = 5 ; a++ ;	a += 4;	
a << 2 ;		

Priorité des opérateurs

” Priorité des opérateurs

```
3 || 0 && 15 + 4 % 3    a + b * c
```

” Parenthésage

```
( 3 || 0 ) && 1    ( 5 + 4 ) % 3    a + ( b * c )
```

- . Pas d'ambiguïté
- . Aide à la relecture du code

” macros

```
#define ABS(a)        if(a < 0) ? 0(a) : (a)
#define SIGN(a)      if(a < 0) ? 0 1 : 1
```

Les fonctions de base

” Fonctions sur les chaînes de caractères

```
x = StrToNum( str);    // renvoie la valeur numérique de str
str = NumToStr( x);    // renvoie la chaîne de caractère composée de l'entier x
i = StrLen( str);     // renvoie la longueur de str (rem pour programmeur C : \0 non compris)
str = StrCat( str1, str2); // str est composé de str1 suivi de str2
SubStr( string, index, length);  StrReplace( string, index, newString); f
```

” Fonctions sur les entiers

```
x = Sin( theta);      // theta en degrés, x : 100 fois le sin, valeur entre -100 et 100
x = Cos( theta);      // theta en degrés, x : 100 fois le cos, valeur entre -100 et 100
theta = Asin( x);     // renvoie le arcsin de 100 x, valeur entre -90 et 90
theta = Acos( x);     // renvoie le arccos de 100 x, valeur entre -90 et 90
x = Sqrt( y);         // renvoie la racine carrée de y
dec = bcd2dec( hexa); // renvoie la valeur en décimal de hexa
```

Les structures algorithmiques

- ~ condition
 - . alternative : **if** (*expression*) *instructions* [**else** *instructions*]
 - . cas_parmi : **switch** (*exp_entiere*)
 - { **case** *valeur1* : *suite_instructions* ; **break** ;
 - case** *valeur2* : *suite_instructions* ; **break** ;
 - default** : *suite_instructions* ; }
- ~ répétition
 - . tant_que : **while** (*expression*) *instructions*
 - . Jusqu'à : **do** *instructions* **while** (*expression*) ;
 - . Jusqu'à : **for**(*instruction*; *expression*; *instruction*) *instructions*
 - . Répéter plusieurs fois : **repeat** (*expression*) *instructions*
- ~ instruction
 - ó *instructions* : *instruction_simple* | *bloc*
 - ó *bloc* : { *suite_instructions* }
 - ó *instruction_simple* : *instruction* ;
 - ó *suite_instructions* : *instruction_simple* *suite_instructions*

La structure alternative

- ~ entier # booléen : 0 : faux, différent de 0 : vrai
- ~ la condition est une expression

```
#define dureeTour 400 // durée de rotation

if
if ( Sensor(S1) )
// le capteur est appuyé
{
  OnFwd(OUT_A, 75); // tourne à droite
  Wait(dureeTour);
  Off(OUT_A); // arrêt
}
if
```

```
#define dureeTour 400 // durée de rotation

if
if ( Sensor(S1) )
// le capteur est appuyé
{
  OnFwd(OUT_A, 75); // tourne à droite
  Wait(dureeTour);
  Off(OUT_A); // arrêt
}
else
// le capteur est relâché
{
  OnFwd(OUT_A, 75); // tourne à gauche
  Wait(dureeTour);
  Off(OUT_C); // arrêt
}
if
```


La structure cas parmi

- ~ pour éviter des conditions en cascade
- ~ l'ordre des valeurs est sans importance, mais les valeurs sont différentes

```

int nb ; // i
i
switch (nb)
{
    case 1 : OnFwd(OUT_A, 75);
             Wait(dureeMarche);
             Off(OUT_A);
             break;
    case 3 : OnFwd(OUT_C , 75);
             Wait(dureeMarche);
             Off(OUT_C);
             break ;
    default :
}
i

```

La structure tant que

- ~ attention aux { }

```

#define dureeAction 100 // durée pour une action d'avancement
#define nbActionsMax 5 // nombre d'actions à faire
int nbActions; // nombre d'actions en cours
i
// initialisation des données pour l'expression
nbActions = 0;

while ( nbActions < nbActionsMax)
{
    OnFwd(OUT_AC , 75);
    Wait(dureeAction);
    Float(OUT_AC);
    nbActions ++;
}
/* il faut s'assurer que l'action modifie les données
afin que l'expression soit fausse à un moment donné */
i

```

Les autres formes de structures de boucle tant que

“ Tâches répétitives

- . tâches de contrôle, de suivi

```
while (Sensor(S3) != valeurRecherchee )
{
    /* traitements */
}
```

“ Boucles infinies

- . tâches de contrôle, de suivi

```
while ( 1 )
{
    /* attente ou vérification */

    /* traitements */
}
```

La structure repeat

“ Nombre d'itérations connues

```
int nbActionsAFaire = 5; // nombre d'actions à faire

repeat(nbActionsAFaire) // repeat(expression)
{
    OnFwd(OUT_AC , 75);
    Wait(dureeAction);
    Float(OUT_AC);
}
```

La structure for

“ Nombre d'itérations connues

```
int nbActionsAFaire = 5; // nombre d'actions à faire

int i ; // compteur

for(i =0; i< nbActionsAFaire ; i++)
{
    OnFwd(OUT_AC , 75);
    Wait(dureeAction);
    Float(OUT_AC);
}
```

Les fonctions NXC

“ Syntaxe

```
[safecall] [inline] typeRetour name(liste_des_arguments)
{
    // instructions
}
```

“ Fonctions

- . 256 fonctions au maximum
 - “ y compris le main et les tâches
- . Pas de déclaration
- . inline
 - “ un appel entraîne la recopie du code de la fonction
 - “ Attention à la taille finale du code !
- . Code non réentrant
 - “ Cf. multitâches

Les fonctions NXC

- “ Entrées
 - . Par valeur : int
 - . Par valeur constante : const int
 - . Par référence : int&
 - . Par référence, valeur constante : const int &

- “ Retour
 - . Type standard
 - . Pas de retour : void

Les fonctions NXC

Passage par valeur : **int**

```
void foo(int x)
{
    x = 2;
}

task main()
{
    int y = 1;           // y est égal à 1
    foo(y);             // y reste égal à 1
}
```

Les fonctions NXC

Passage par valeur constante : `const int`

```
void foo(const int x)
{
    int z;
    z = x;    // ok
    x = 1;    // erreur car x ne peut pas être modifié (const)
}

task main()
{
    int y;
    foo(2);   // ok
    foo(4*5); // ok - l'expression est une constante
    foo(y);   // erreur, y n'est pas une constante
}
```

Les fonctions NXC

Passage par référence : `int &`

```
void foo(int &x)
{
    x = 2;
}

task main()
{
    int y = 1; // y est égal à 1
    foo(y);   // y est maintenant égal à 2
    foo(2);   // erreur, seules les variables ont des références
}
```

Passage par référence constante, peu utile: `const int &`

Les directives de compilation

É **#define**

- . Macro substitution avant compilation
- . Définition d'une variable de compilation
- ó **#undef** : fin de la définition, fin de la macro substitution

É **#include**

- . Inclusion de fichier

```
#include "foo.nxc" // ok
```

```
#include <foo.nxc> // erreur en NXC
```

É **#if, #ifdef, #ifndef, #else, #endif**

- . Directives de compilation

Inclusion multiples de fichiers

- “ Directive pour assurer l'inclusion unique d'un fichier
 - . Pas de multiples déclarations

```
#ifndef FILENAME
#define FILENAME

#include " fichiersUtilises "

// instructions NXC

#endif
```

Plan

- “ Caractéristiques du robot NXT de Lego®
- “ Programmation du robot NXT
 - . Gestion des entrées/sorties
- “ Not eXactly C : pas exactement du C
 - . Langage de programmation proche du langage C
 - . Programmation des briques NXT de Lego ®
- “ Programmation multi-tâches en NXC
 - . Temps, tâches et synchronisation

Les tâches

- “ Toujours une tâche principale
 - . `task main ()`
 - . Lancée au début de l'application
- “ Nombre de tâches limité par la plateforme
 - . 256 tâches ou fonctions (y compris le main)
- “ Définition d'une tâche
 - . Pas de paramètres
 - . Pas de retour

```
task maTache ()  
{  
    /* séquence de commandes */  
}
```

Gestion des tâches

~ Lancement/arrêt de tâches

```
ó start taskName;
ó Precedes (taskName1, taskName2);
ó stop taskName;
```

```
task readSensor()
{
    while( 1 )
    {
        if ( Sensor(S1) )
        {
            // stop controlRobot;
        }
    }
}

task controlRobot()
{
    í
}
```

```
task main ()
{
    SetSensorTouch(S1);
    Precedes( controlRobot, readSensor);
}
```

jean-philippe.babau@univ-brest.fr

Tâches régulières

~ Maîtrise des dates d'activation des actions

- . Scrutation périodique
- . Commande régulière

~ Limitation de l'occupation du processeur

```
#define ATTENTE 1000 // période d'attente égale à 1 seconde

task readSensor()
{
    while( 1 )
    {
        Wait(ATTENTE ); // mise en sommeil de la tâche pour ATTENTE * 1 ms

        position = Sensor(S1) ; // acquisition
        í
    }
}
```

jean-philippe.babau@univ-brest.fr

UBO

Gestion du temps

- “ Tick
 - . Précision : 1ms
- “ Wait(timeout)
 - . Attente en ms
 - . Paramètre : **unsigned int** (environ 1 minutes de 0 à 65535)
 - . **Attention** : pas de valeur négative, sinon attente longue 0
- “ Primitives
 - x = CurrentTick(); // renvoie le temps en ms, type unsigned long (4 octets) (environ 50 jours)
 - x = FirstTick(); // renvoie le temps en ms où le programme a été lancée
- “ Arrêt programmé de la brique
 - SleepTimer(); // renvoie la valeur du compteur d'arrêt de la brique
 - SetSleepTime(tps); // initialise la valeur du compteur d'arrêt automatique à tps minutes
 - SleepTime(); // renvoie le temps courant restant avant l'arrêt automatique
 - ResetSleepTimer(); // remise à 0 du compteur d'arrêt automatique (évite l'arrêt)
 - SetSleepTimer(tps); // initialise le compteur courant d'arrêt automatique à tps minutes
 - SleepNow(); // arrêt de la brique

jean-philippe.babau@univ-brest.fr

UBO

Partage de données

- “ Partage d'une variable entière
 - . Déclaration globale
- “ Partage par mutex

```
int x ; // variable partagée entre T1 et T2

task T1()
{
  ...
  x = a - b; // opération non atomique
  ....
}

task T2()
{
  ...
  y = x ; // la valeur de x, et donc de y, n'est pas garantie
  ....
}
```

```
int x ; // variable partagée entre T1 et T2
mutex prot_x ; // mutex de protection de x

task T1()
{
  ...
  Acquire(prot_x );
  x = a - b; // opération protégée
  Release(prot_x );  ....
}

task T2()
{
  int y ;  ....
  Acquire(prot_x );
  y = x ; // la valeur de x est garantie
  Release(prot_x );  ....
}
```

jean-philippe.babau@univ-brest.fr

UBO

Déclenchement d'une tâche

```
#define ATTENTE 100

task readSensor()
{
    start controlRobot;

    while( 1 )
    {
        if (Sensor(S1))
        {
            Wait(ATTENTE);

            start controlRobot;
        }
    }
}

task controlRobot()
{
    f
}

task main ()
{
    SetSensorTouch(S1);
    start readSensor;
}
```

relance la tâche au début

jean-philippe.babau@univ-brest.fr

UBO

Scrutation des capteurs


```
#define ATTENTE 100 //tester 10, 100, 1000, 5000

task switchMotor()
{
    bool running = true;
    OnFwd(OUT_A, 75);
    while( true )
    {
        if (Sensor(S1) && running)
        {
            running = false;
            Off(OUT_A);
            Wait(ATTENTE );
        }
        if (Sensor(S1) && !running)
        {
            running = true;
            OnFwd(OUT_A, 75);
            Wait(ATTENTE );
        }
    }
}

task main ()
{
    SetSensorTouch(S1);
    start switchMotor;
}
```

jean-philippe.babau@univ-brest.fr

Ordonnancement des tâches

- “ Ordonnancement par partage de temps
- “ Comportement du moteur A 
- “ Attention à ne pas multiplier les tâches
 - . Partage et cohérence de données
 - . Synchronisation des actions

```
task main()
{
    Precedes(Go , Commande , Arret);
}
```

```
int go = 0 ; // déclenchement des moteurs
```

```
task Commande()
{
    if (go == 1)
    {
        OnFwd(OUT_A,75);
    }
}
```

```
task Arret()
{
    if (go == 0)
    {
        Off(OUT_A);
    }
}
```

```
task Go()
{
    go = 1;
}
```

Mise en place de tâches

- “ Conception
 - . Tâche de démarrage
 - “ Initialisations et activations des autres tâches
 - . Tâches de réaction à un événement externe ou un événement programmé
 - “ Un nouveau message
 - “ Une nouvelle mesure (capteurs)
 - “ Une alarme programmée sur un dépassement de timeout
 - . Tâches régulières
 - “ Scrutation de capteurs
 - “ Gestion des actionneurs
 - “ Suivi du fonctionnement (*monitoring*)
- “ Programmation
 - . main
 - . Tâches régulières
 - “ while(1) { Wait(Attente); Actions } ou while(1) { Actions Wait(Attente); }
 - . Tâches « en continu »
 - “ while(1) { Actions }
 - . Tâches activées par une autre tâche
 - “ start taskName;

Conclusion

- “ Robot
 - . Langage de programmation de haut niveau
 - “ Langage interprété
 - . Communications
 - “ Capteurs : l'utilisation dépend de la configuration
 - “ Moteurs : commande en vitesse, en position (PID)
 - “ IHM : 3 boutons et un écran (8 lignes x 15 colonnes, graphique 100x64 pixels), générateur de sons
 - “ Gestion de fichiers (4 au maximum et taille limitée)
 - “ Communication bluetooth (1 Master/3Slaves)
 - “ Batterie : suivi
- “ Programmation
 - . NXC
 - “ pas de pointeurs
 - “ fonctions non réentrantes
 - . Présentation du code
 - “ commentaires, indentation, choix des noms de variables, définition des constantes
- “ Multitâches
 - . Précision du temps : 1ms
 - . Mise en place des tâches : main, tâches régulières, tâches continues
 - . Attention au partage des données et à la synchronisation : utilisation du mutex

Références utilisées pour faire ce cours

« Not eXactly C (NXC) Programmer's Guide » Version 1.0.1 b33, by John Hansen, Oct 2007

<http://bricxcc.sourceforge.net/nxc/>

« Programming Lego NXT Robots using NXC » Version 2.1, by Danielle Benedetti, April 2007

<http://people.cs.uu.nl/markov/lego/>

« LEGO ® MINDSTORMS® NXT » Hardware Developer Kit

http://www.cse.unr.edu/~monica/Courses/CPE470-670/Resources/User_Manual.pdf