Programmation NQC
Application à la commande d'un robot RCX 2.0 Lego©

Richard MOREAU

Laboratoire Ampère - Département GMC

richard.moreau@insa-lyon.fr

Plan du cours

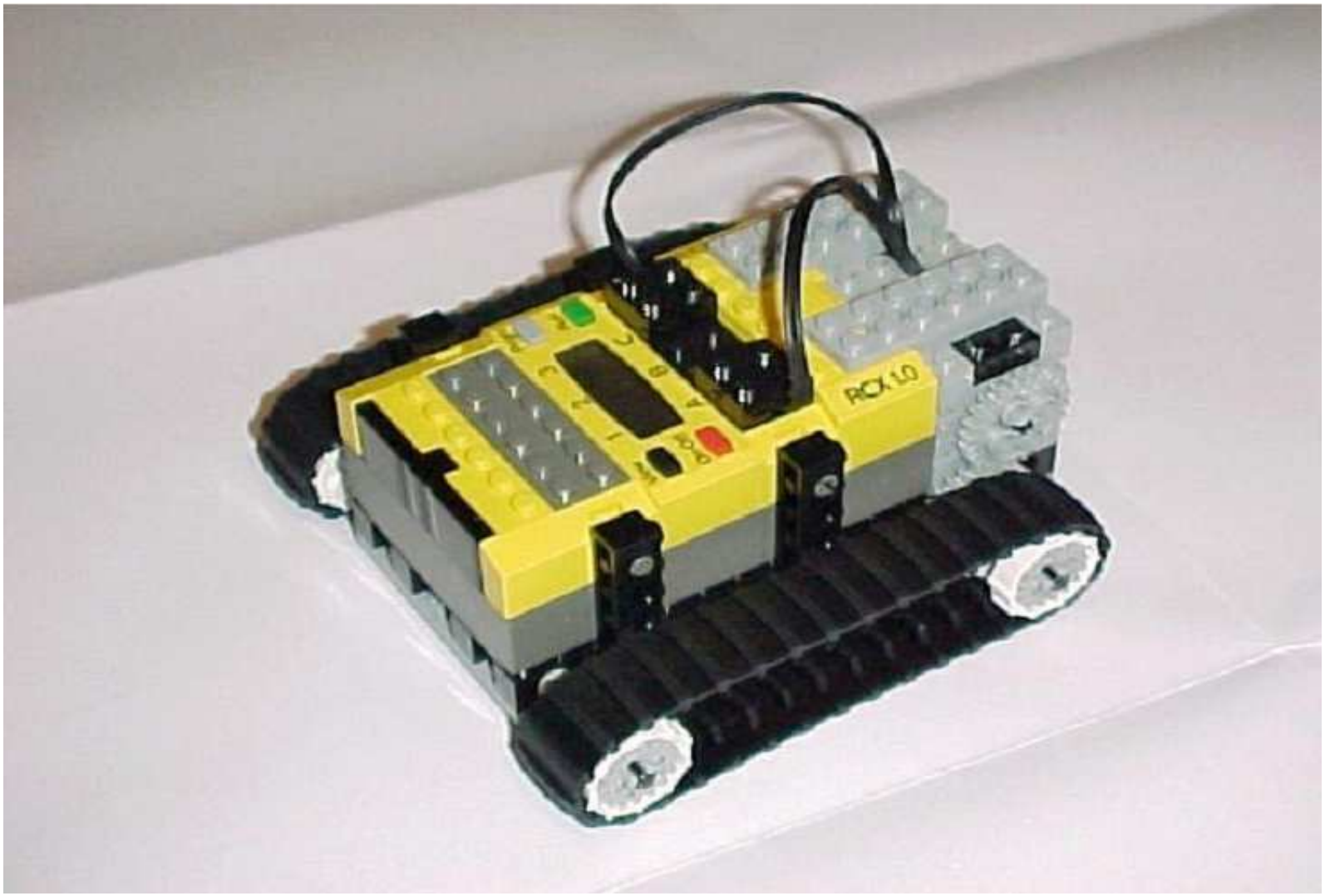
1. Caractéristiques du robot RCX 2.0 de Lego©
2. Programmation du Robot RCX 2.0
3. Not Quite C : “Pas exactement du C”
4. Programmation multi-tâches en NQC

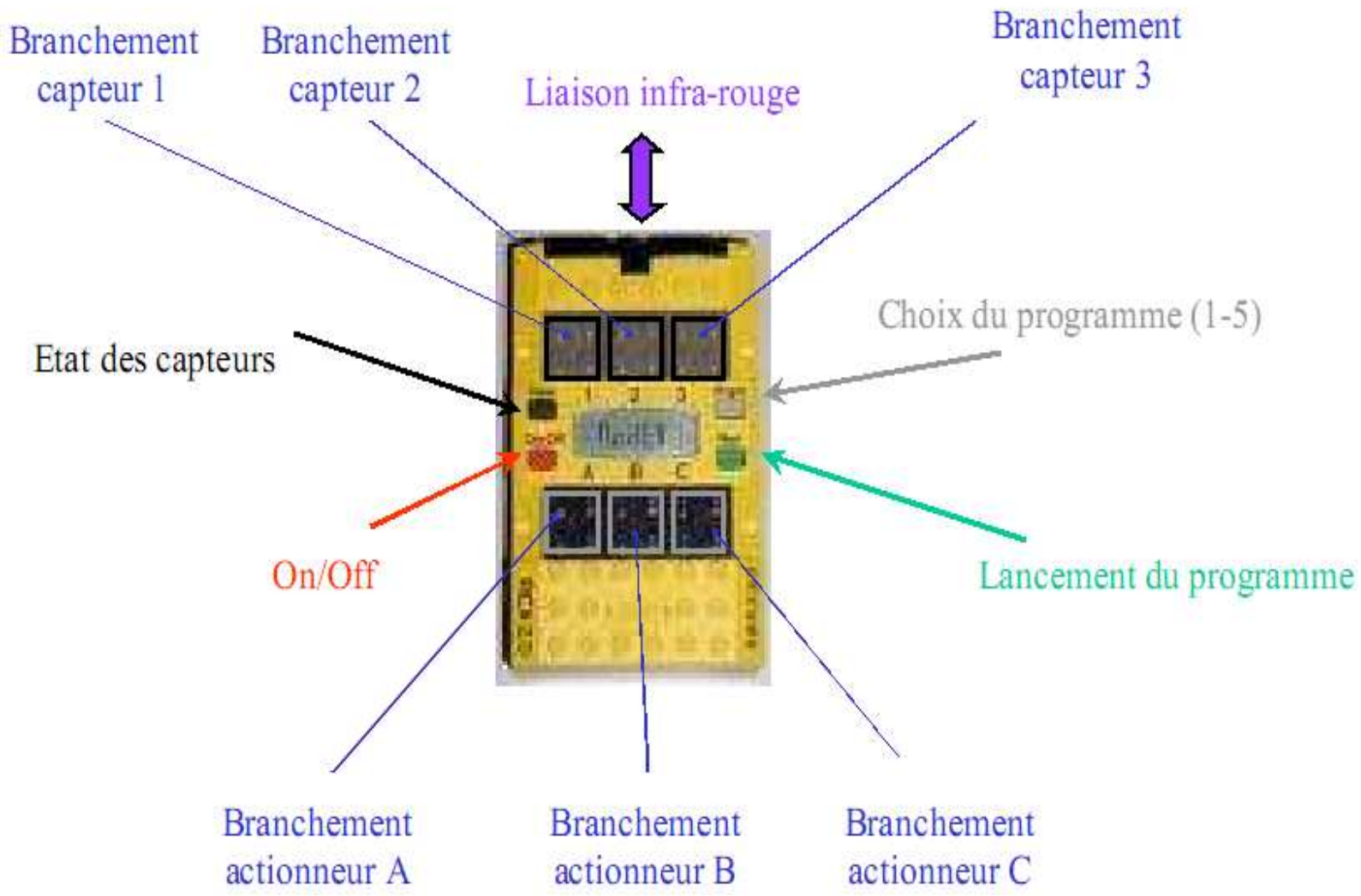
Objectifs du cours

- Commande d'un robot RCX 2.0 (Lego Mindstorms©)
  - Trois capteurs (deux capteurs de contact et un de luminosité)
  - Trois actionneurs (deux moteurs)
  - Un générateur de son
  - Un écran LCD
  - Un médium infra-rouge
- Structure d'un programme NQC
  - main, données, algorithmes, fonctions, subroutines
- Structure d'un programme multi-tâches NQC
  - Tâches, synchronisation

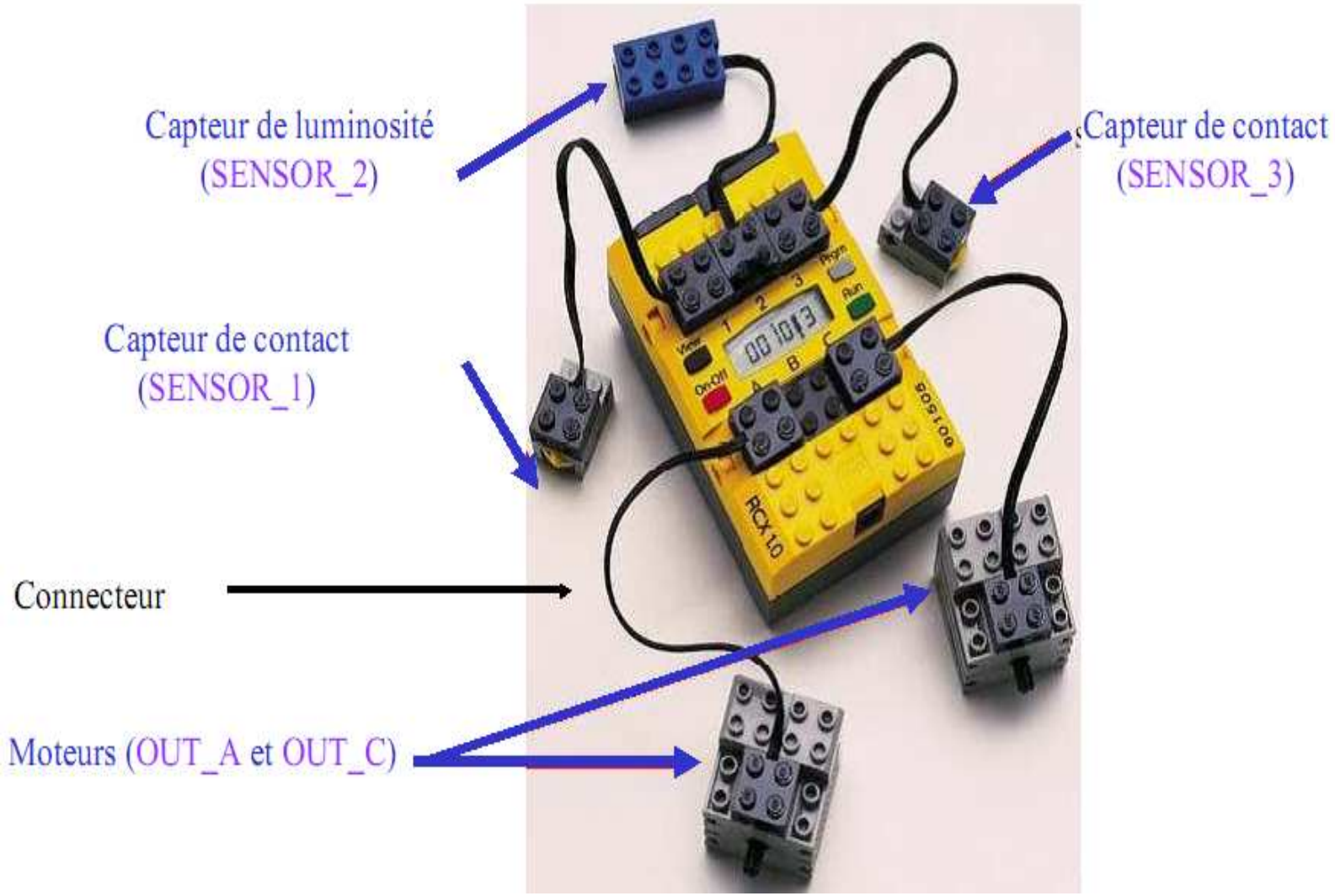
Plan du cours

Robot RCX 2.0





RCX 2.0 avec ses capteurs et ses actionneurs



Architecture matérielle et logicielle

- Matériel
  - Micro-contrôleur HITACHI H8300
    - 16 Mhz, 16 ko ROM et 32 ko de RAM
    - 6 ko de RAM disponible pour l'application
- Logiciels
  - Bootcode
    - Initialisation de la carte
    - Gestion des entrées/sorties
    - Communication Infra-rouge (IR)
  - Firmware
    - API du Lego©
    - Services d'accès aux entrées et aux sorties
    - Exécutif multi-tâches
    - Par défaut : code interprété (fichier FIRM0328.LGO)
    - Autres firmwares : leJOS (Java), BrickOS ou legOS (C/C++)
- Mémoires
  - ROM : mémoire persistante
  - RAM : mémoire volatile (persistante si alimentée)

Programmation

- Développement croisé
  - Machine hôte (PC, Mac) : développement et mise au point
  - Machine cible (RCX 2.0) : exécution d'un programme
  - Liaison infra-rouge : téléchargement et communications
- Environnement de développement (*Bricx Command Center* ou *BricxCC*)
  - Éditeur de langage NQC
  - Compilation (*Compile*) et téléchargement (*Download*)
  - Suivi des variables (*Watch*)
  - Diagnostic de la carte
- Simulation
  - Simulateur web SIMBOT
  - Émulateur (emulegOS basé sur legOS)

Quelques problèmes

- Le robot n'est pas visible par le logiciel
  - Penser à utiliser le bon port USB
  - Allumer le robot !
- Plus de piles
  - Penser à re-télécharger le firmware
- Lumière artificielle
  - Interférences avec la liaison IR
- Téléphones, PDA, ...
  - Interférences avec la liaison IR
- Interférences entre la liaison IR et le capteur de luminosité
  - Placer le capteur à l'opposé de la zone de communication

Plan du cours

Configuration des capteurs

- 3 capteurs numérotés 0, 1 et 2 :
  SENSOR_1, SENSOR_2, SENSOR_3
- Un type

SENSOR_TYPE_NONE	Capteur générique passif
SENSOR_TYPE_TOUCH	Capteur de contact
SENSOR_TYPE_TEMPERATURE	Capteur de température
SENSOR_TYPE_LIGHT	Capteur de luminosité
SENSOR_TYPE_ROTATION	Capteur de rotation

- 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
SENSOR_MODE_FAHRENHEIT	Degrés Fahrenheit
SENSOR_MODE_CELSIUS	Degrés Celsius
SENSOR_MODE_ROTATION	Rotation (16 ticks par tour)

Une configuration : un type + un mode

Configuration	Type	Mode
SENSOR_TOUCH	SENSOR_TYPE_TOUCH	SENSOR_MODE_BOOL
SENSOR_LIGHT	SENSOR_TYPE_LIGHT	SENSOR_MODE_PERCENT
SENSOR_CELSIUS	SENSOR_TYPE_TEMPERATURE	SENSOR_MODE_CELSIUS
SENSOR_FAHRENHEIT	SENSOR_TYPE_TEMPERATURE	SENSOR_MODE_FAHRENHEIT
SENSOR_PULSE	SENSOR_TYPE_TOUCH	SENSOR_MODE_PULSE

Action des capteurs

- Initialiser le type, le mode et la configuration
  - SetSensorType(sensor,type);

    SetSensorType(SENSOR_1,SENSOR_TYPE_TOUCH)
  - SetSensorMode(sensor,mode);

    SetSensorMode(SENSOR_1,SENSOR_MODE_RAW)
  - SetSensor(sensor,configuration);

    SetSensor(SENSOR_1,SENSOR_TOUCH)
- Récupérer une information d'un capteur
  - x=SENSOR_1; Lit sensor_1 et sauvegarde la valeur dans x
  - SensorValue(n); avec n = 0, 1 ou 2 selon le capteur à lire

    par exemple x= SensorValue(0) fait la même action que x=SENSOR_1;
- Remettre à 0 un capteur cumulatif
  - ClearSensor(sensor);

    ClearSensor(SENSOR_1);

Configuration des actionneurs

- 3 actionneurs (numérotés 0, 1 et 2) :
  - OUT_A, OUT_B, OUT_C
  - OUT_A + OUT_B
- Un Mode
  - OUT_OFF
  - OUT_ON
  - OUT_FLOAT arrêt en douceur
- Une direction (actif si OUT_ON)
  - OUT_FWD
  - OUT_REV
  - OUT_TOGGLE Changement de direction (avant->arrière ou arrière->avant)
- Un niveau (actif si OUT_ON)
  - 0 (OUT_LOW) à 7 (OUT_FULL)
  - OUT_HALF
- Par défaut
  - OUT_FULL, OUT_FWD

Action des actionneurs

- Configuration
  - SetDirection(outputs,direction); SetDirection(OUT_A,OUT_FWD);
- Mise au point du niveau
  - SetPower(outputs,niveau); SetPower(OUT_A,OUT_FULL);
- Arrêt / Marche
  - SetOutput(outputs,mode); SetOutput(OUT_A,OUT_ON);
- Appels de haut niveau
  - On(outputs); Off(outputs); On(OUT_A);
  - Fwd(outputs); Rev(outputs); Toggle(outputs);
  - OnFwd(outputs); OnRev(outputs); OnFwd(OUT_A);
  - OnFor(outputs,time); OnFor(OUT_A,400);

Le son

- Un générateur de son
- Faire un son
  - PlaySound(sound);
    sound : SOUND_CLICK, SOUND_DOUBLE_BEEP, ...
- Une fréquence
  - PlayTone(frequency,duration); PlayTone(440,50);
- Arrêter / Reprendre
  - MuteSound(); UnmuteSound();

L'affichage *via* l'écran LCD

- Un mode

DISPLAY_WATCH	affiche l'heure (par défaut)
DISPLAY_SENSOR_1	affiche la valeur du capteur 1
DISPLAY_SENSOR_2	affiche la valeur du capteur 2
DISPLAY_SENSOR_3	affiche la valeur du capteur 3
DISPLAY_OUT_A	affiche les paramètres de l'actionneur A
DISPLAY_OUT_B	affiche les paramètres de l'actionneur B
DISPLAY_OUT_C	affiche les paramètres de l'actionneur C
DISPLAY_USER	mode programmable

- Configurer un mode
  - SelectDisplay(mode);
- Affiche une valeur (en mode DISPLAY_USER)
  - SetUserDisplay(value,precision);
    SetUserDisplay(1234,2); affiche 12.34

- Datalog
- Configurer la zone de sauvegarde
  - CreateDatalog(n);
    zone de stockage de n valeurs (n_{max}=1000)
- Sauvegarde
  - AddToCatalog(value);
    - AddToCatalog(x);
    - AddToCatalog(Timer(0));
    - AddToCatalog(SENSOR_1);
- Récupération des données sur la machine de développement avec le logiciel BricxCC

Le service de communication

- Messagerie
  - Communication d'entiers entre 0 et 255
  - 0 réservé pour détecter l'absence de nouveau message
- Émission
  - SendMessage(value);
    - SendMessage(3); envoie 3
- Réception
  - x=Message();
- Effacement du buffer de réception
  - ClearMessage();

Autres fonctions

- Générateur aléatoire
  - Random(n); renvoie une valeur comprise entre 0 et n
- Niveau de la batterie
  - BatteryLevel(); valeur en millivolts
- Autres services
  - SelectProgram(3); lancement du 4ème programme
  - SetWatch(16,0); Mise à l'heure 16h00
  - h=Watch(); Récupération de l'heure en minutes

Programme de commandes

- Une tâche principale
  - task main
  - Séquence de commandes

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

Plan du cours

1. Caractéristiques du robot RCX 2.0 de Lego©
2. Programmation du Robot RCX 2.0
3. Not Quite C : "Pas exactement du C"
4. Programmation multi-tâches en NQC

- Déclaration de données
  - Initialisation des données
  - Définition des constantes
- Tâche : séquences d'instructions
  - Commande robot
  - Opérations sur des données
  - Appels de fonctions
  - Structures algorithmiques

- Not Quite C : c'est presque du C
- Langage de programmation proche du langage (Syntaxe C-like)
- Programmation des briques RCX de Lego©
- Programmation multi-tâches

Syntaxe

- Différence entre les minuscules et les majuscules
- Liste de mots clés réservés (cf. diapo suivante)
- Identificateur : suite de lettres, chiffres, _ ne commençant pas par un chiffre
- Commentaire

  /* commentaire à la C */

  // commentaire à la C++

Les données

- Toute information manipulée par le programme (constantes, variables)
- Caractérisée par : un identificateur et un commentaire
- Deux types :
  - entier signé sur 16 bits (2 octetx, [-32768, 32767]) : **int**
  - pointeur d'entier : **int***
- Tableau
  - Tableaux d'entiers
  - Doit être initialisé explicitement
  - Ne peux pas être passé en argument d'une fonction

Les données

- Valeurs
  - Décimal
  - Hexadécimal
  - Booléen : 0 correspond à *false* et différent de 0 correspond à *true*
- Déclaration de données
  - **Int** nombreTour; // nombre de tours effectués
  - **Int** MissionTerminee=0;
  - **Int** a, b;
- Déclaration de constantes
  - # **define** Annee 2008 // Année courante à modifier
- Déclation de tableaux
  - **int** lesPositions[10]; // ensemble de points à atteindre
  - Déclaration de lesPositions[0], lesPositions[1], ..., lesPositions[9]

Liste non exhaustive de mots clés

do	int	sub	break
else	monitor	switch	task
case	false	repeat	true
catch	for	return	void
const	goto	sign	while
continue	if	start	stop

Les opérateurs

- Liste d'opérateurs
  - Opération : + - * / %(modulo)
  - Incrémentation / décrémentation : ++ / –
  - Affectation : =
  - Comparaison : == != < > <= >=
  - Logique : ! (négation) && (et) || (ou)
  - Attention à la priorité des opérateurs
- Exemples
  - 5/4 5%4
  - 5==4 7<=19
  - !5 5&&2
  - int a=5; a++;

Les structures algorithmiques

- Condition
  - Alternative : **if** (*expression*) *instructions* [**else** *instructions*]
  - Cas Parmi : **switch**(*exp_entière*)
    { **case** *valeur1* : *suite_instruction*; **break**;
    case *valeur2* : *suite_instruction*; **break**;
    defaut : *suite_instruction* ; }
- Répétition
  - Tant que : **while** (*expressions*) *instructions*
  - jusqu'à : **do** *instructions* **while** (*expression*) ;
  - Répéter plusieurs fois : repeat (*expression*) *instructions* ;
- instructions
  - instructions : *instruction_simple*
  - bloc : *suite_instructions*

La structure alternative

Booléen : 0 => faux, différent de 0 => vrai

La condition est une expression

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


if (SENSOR_1) // si le capteur est appuyé
{
	OnFwd(OUT_A+OUT_C); // avance
	Wait(dureeTour);
	Off(OUT_A+OUT_C); // arrêt
}
else // Le capteur est relâché
{
	OnRev(OUT_A+OUT_C); // recule
	Wait(dureeTour);
	Off(OUT_A+OUT_C); // arrêt
}
```

La structure cas parmi

Pour éviter des conditions en cascade

Peu importe l'ordre des valeurs mais les valeurs sont différentes

```
int nb // déclaration variable
# define dureeMarche 400 // durée de fonctionnement
switch (nb) // si le capteur est appuyé
{
        Case 1 :
                OnFwd(OUT_A); // tourne à droite
                SelectDisplay(DISPLAY_OUT_A);
                Wait(dureeMarche);
                Off(OUT_A); // arrêt
                break;
        Case 3 :
                OnFwd(OUT_C); // tourne à gauche
                SelectDisplay(DISPLAY_OUT_C);
                Wait(dureeMarche);
                Off(OUT_C); // arrêt
                break;
        default :
                SelectDisplay(DISPLAY_USER);
                SetUserDisplay(0,2);
}
```

La structure tant que

Pas de booléen, 0 : faux, différent de 0 : vrai

La condition est une expression

attention aux

```
# define dureeAction 100 // durée de fonctionnement
# define nbActionsMax 5 // Nombre d'actions à réaliser
int nbActions // Nombre d'actions en cours

nbActions = 0 // Initialisation des données pour l'expression

while (nbActions <nbActionsMax)
{
        OnFwd(OUT_A+OUT_C); // avance
        Wait(dureeAction);
        Off(OUT_A+OUT_C); // arrêt
        nbActions++;
}
/* Il faut s'assurer que l'action modifie les données

afin que l'expression soit fausse à un moment donné */
```

Les autres formes de structures de boucle tant que

- Tâches répétitives

```
while (SENSOR_1 != ValeurRecherchée)
{
        // Traitement
}
```

- Boucles infinies

```
while (1)
{
        // Attente ou vérification

        // Traitement
}
```

La structure Repeat

Nombre d'itérations connues

```
# define dureeAction 100 // durée de fonctionnement
int nbActionsAFaire = 5 // Nombre d'actions à faire

repeat(nbActionsAFaire)
{
        OnFwd(OUT_A+OUT_C); // avance
        Wait(dureeAction);
        Off(OUT_A+OUT_C); // arrêt
}
// Il existe un programme équivalent avec un seul appel : OnFor(outputs,time)
```

- Fonctions “inline”
  - Un appel entraîne la recopie du code de la fonction
  - Attention à la taille finale du code
- Entrées
  - Par valeur : **int**
  - Par valeur constante : **const int**
  - Par référence : **int&**
  - Par référence, valeur constante : **const int &**
  - Par pointeur : **int***
  - Par pointeur constant : **const int***
- Pas de retour en NQC : **void**
- Syntaxe

```
void name(liste_des_arguments)
{
    // instructions
}
```

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
}
```

Passage par valeur constante : **const int**

```
void foo(const int x)
{
        PlaySound(x); // ok
        x=2; // erreur car x ne peut pas être modifié (x est une constante)
}

task main()
{
        foo(2); // ok
        foo(4*5); // ok
        foo(x); // erreur, x n'est pas une constante
}
```

Les Subsoutines NQC

- Fonctions
  - Un appel : pas de recopie du code de la subroutine
  - Non réentrantes
- Restrictions
  - 8 subroutines au maximum
  - Pas d'arguments
  - Pas d'appel à une autre subroutine
  - Par pointeur constant : **const int***
- Syntaxe

```
sub name()
{
        // instructions

        // pas d'appel à une autre subroutine
}
```

Les directives de compilation

- #define
  - Macro substitution avant compilation
  - Définition d'une variable de compilation
  - #undef : fin de la définition, fin de la macro substitution
- #include
  - Inclusion de fichier :

```
#include "foo.nqh" // ok
#include <foo.nqh>// erreur en NQC
```

- #if, #ifdef, #ifndef, #else, #endif
  - Directives de compilation
- #pragma reserve **startWord endWord**
  - Réservation d'espace mémoire
  - L'utilisation de 3 compteurs (counter) nécessite l'ajout de la directive :
    #pragma reserve 0 1 2

Inclusion multiples de fichiers

Directive pour une seule inclusion

```
#ifndef FILENAME
#define FILENAME

        #include “fichiersUtilisés”
        // Instructions NQC
#endif
```

Plan du cours

1. Caractéristiques du robot RCX 2.0 de Lego©
2. Programmation du Robot RCX 2.0
3. Not Quite C : “Pas exactement du C”
4. Programmation multi-tâches en NQC

Gestion du temps

- Caractéristiques des timers
  - Le nombre de timers dépend de la cible
  - précision : 100 ms (10 ticks par seconde)
- Action sur les timers
  - ClearTimer(**n**); // reset du timer **n** à 0
  - Timer(**n**); ou x=Timer(**0**); // retour de la valeur en dixième de secondes
- Attente
  - Wait(**NombreDeTicks**) ; // en centième de secondes
  - Wait(**100**); // attente d'une seconde

Les tâches

- Toujours une tâche principale
  - task main
  - Lancée au début d'application
- Nombre de tâches limité par la plateforme (10)

- Lancement / arrêt de tâches
  - start **taskName**;
  - stop **taskName**;

```
task main()
{
	SetSensor(SENSOR_1, SENSOR_TOUCH);
	start controlRobot;
	start readSensor;
}
task controlRobot()
{
// instructions
}
task readSensor()
{
		while(1)
		{
		  if (SENSOR_1 == 1)
		  {
		      stop controlRobot;
		  }
		}
}
```

Problème de synchronisation : redémarrage d'une tâche


```
task main()
{
	int running = 1;
	start controlRobot;
	while(1)
	{
		if ((SENSOR_1 == 1) && (running == 1))
		{
		  running == 0;
		  stop controlRobot;
		  Wait(200);
		}
		if ((SENSOR_1 == 1) && (running == 0))
		{
		  running == 1;
		  start controlRobot;
		  Wait(200);
		}
	}
}
```

Synchronisation par sémaphore

- Sémaphore : variable entière
  - int **semTrigger** = 0 ; sémaphore de synchronisation
  - int **semMutex** = 1 ; sémaphore de protection
- Déclenchement d'évènement
  - semTrigger = 0 ;
- Attente d'évènement
  - until (**semTrigger**) = 1) ;
  - **semTrigger** = 0 ;
- Prise de sémaphore / .. / libération de sémaphore
  - until(**semMutex** = 1) ;
  - **semMutex** = 0 ;
  - **semMutex** = 1 ;

Tâche périodique

- Maîtrise des dates d'activation des actions
  - Scrutation périodique
  - Commande régulière
- Limitation de l'occupation du processeur

```
#define PERIODE 100 // période de scrutation égale à 1 seconde
task readSensor()
{
	while(1)
	{
		Wait(PERIODE);
		position=SENSOR_1; // acquisition
		...
	}
}
```

Partage de données

- Partage d'un variable entière
  - Déclaration globale
- Appels atomiques
  - Uniquement des opérations d'affectation /consultation sur un entier

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

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

task T2()
{
        ...
        y=x // La valeur de x n'est pas garantie
        ...
}
```

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

task T1()
{
        int x1; // variable locale à T1
        x1=a-b; // opération locale
        x=x1; // opération atomique
}

task T2()
{
        ...
        y=x // La valeur de x est garantie
        ...
}
```

Les événements : configuration

- Événement capteur

```
#define EVENT_A 0;
SetSensor(SENSOR_1, SENSOR_TOUCH);
SetEvent(EVENT_A, SENSOR_1,EVENT_TYPE_PRESSED);
```

- Événement timer

```
#define EVENT_B 1;
ClearTimer(0);
SetEvent(EVENT_B,Timer(0),EVENT_TYPE_HIGH);
SetUpperLimit(EVENT_B,10);
```

Les événements : les actions

- Suivi : monitor(masque) et catch(masque)

```
monitor(EVENT_MASK(EVENT_A)|EVENT_MASK(EVENT_B)|EVENT_MASK(4))
{
	Wait(1000);
}
catch(EVENT_MASK(EVENT_B))
{
	PlaySound(SOUND_DOWN)); // L'événement Event_B arrive
	ClearTimer(0); // Remise à 0 du timer 0
}
catch
{
	PlaySound(SOUND_UP)); // l'événement Event_A ou l'événement 4 arrive
}
```

Contrôle d'une tâche


```
task readSensor()

{
    start controlRobot;
    Wait(200);
    running=1;
    while(1);
    {
        if((SENSOR_1==1) && (running==1))
        {
            running=0;
            Wait(200);
        }
        if((SENSOR_1 ==1) && (running==0))
        {
            running=1;
            Wait(200);
        }
    }
}
```

```
int running = 0;
task main()

{
    SetSensor(SENSOR_1, SENSOR_TOUCH);
    start readSensor;
}

task controlRobot()
{
    while(1)
    {
        until(running=1);
        ... // Actions 1
        until(running=1);
        ... // Actions 2
        until(running=1);
        ... // Actions 3
    }

}
```

Les événements : configuration

- 16 événements configurables (numérotés de 0 à 15)
- Configuration : source de l'événement + type d'événement
  SetEvent(**numero**,**source**,**type**);
- Annulation d'un événement
  ClearEvent(**numero**); ClearAllEvents();
- Définitions d'intervalles
  SetLowerLimit(**numero**,**low_limit**);
  SetUpperLimit(**numero**,**upper_limit**);
- Sources de l'événement
  Capteur, message buffer, timer, counter
- Types d'événement : critère d'émission de l'événement
  - Capteurs : EVENT_TYPE_PRESSED,EVENT_TYPE_RELEASED, ...
  - Intervalles : EVENT_TYPE_LOW,EVENT_TYPE_NORMAL, EVENT_TYPE_HIGH
  - Message : EVENT_TYPE_MESSAGE

Mise en place de tâches 1/2

- 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*)

Mise en place de tâches 2/2

- Programmation
  - Main
  - Tâches périodiques
    - while(1)Wait(Periode); *Actions*
    - while(1)*Actions*; Wait(Periode)
  - Tâches “en continu”
    - while(1)*Actions*
  - Tâches activés par une autre tâche
    - start *taskName*
  - Programmation événementielle
    - Un événement -> une tâche

Conclusion

- Robot
  - Langage de programmation haut niveau:
    - Langage interprété
  - Capteurs, moteurs, écran LCD, son, mémoire de stockage, batterie, communication IR
    - Capteurs : l'utilisation dépend de la configuration
    - Moteurs : commande de "-7" à "+7"
    - Mémoire limitée à 1000 entiers
    - Communication série half duplex, une trame = un entier
- Programmation
  - NQC
    - Limité aux entiers, pas de fonctions classiques mais plutôt des macros
  - Présentation du code
    - Commentaires, indentation, choix des noms de variables, définition des constantes
- Multi-tâches
  - Précision du temps : 10 ms
  - Mise en place des tâches (10 maximum)
  - Choix des types de tâches (main, tâches périodiques, tâches continues)
  - Attention au partage des données et à la synchronisation

Exemple

```
#define EVENT_A 0;

task main()
{
    SetSensor(SENSOR_1, SENSOR_TOUCH);
    SetEvent(EVENT_A, SENSOR_1,EVENT_TYPE_PRESSED);

    while(1)
    {
      monitor(EVENT_MASK(EVENT_A))
      {
        while(1); // ou while(1) Wait(100);
      }
      catch(EVENT_MASK(EVENT_A))
      {
        PlaySound(SOUND_CLICK);
      }
```

Références pour ce cours

- “NQC Programmer’s Guide, version 3.1 r5”
  by Dave BAUM & John HANSEN
- “Programming Lego Robots”
  by Mark Overmars
- “Programmation NQC”
  by Prof. Jean Philippe BABAU
- “Introduction to Computers and Programming”
  by Prof. I. K. LUNDQVIST

http://www.mit.edu/ kristina/Lego/UnifiedF05/Lectures/Lecture1.pdf