Alternative Programming Languages

LMICSE Workshop
June 14 - 17, 2005
Alma College
Available Languages for the RCX

- Mindstorms ships with its own iconic programming language, but this is a very limited language.
- So other have developed for Mindstorms almost all the usual programming languages, including:
  - C
  - C++
  - Java
  - Lisp
  - Ada
  - Forth
  - Smalltalk
Primary RCX Programming Environments

- LejOS
  - Java, replacement firmware
- NQC (Not Quite C)
  - A simplified C-based language which uses the standard firmware
- BrickOS (formally known as LegOS)
  - C and C++, replacement firmware
- RCXLisp
  - Common Lisp, replacement firmware
LejOS

- Since LejOS is the workshop’s language/environment of instruction, we will not address it during this session
- But you can find the LejOS home page at
NQC

- Well suited for those new to programming
- Often used in non-majors courses
- Allows the demonstration of most non-object oriented aspects of introductory programming
- Easy introduction to concurrent programming issues
NQC Basics

- C-like syntax
- Limited by use of standard firmware
  - 10 threads
  - 32 global variables, 16 local variables per thread
  - Integers only
  - Byte codes are interpreted
  - No call stack, no memory management
- Functions
  - pass by value and pass by reference parameters
  - all function return void
- Simple event model
- Limited arrays
Recall that there are three power outputs, and in NQC the motors attached to them are called

- OUT_A
- OUT_B
- OUT_C

Basic motor control commands

- OnFwd( ... ) : run the ... motor(s) forward
- OnRev( ... ) : run the ... motor(s) in reverse
- Off( ... ) : stop the ... motor(s)

Another useful command

- Wait( ... ) : pause the program execution for ... microseconds
task main () {
    OnFwd(OUT_A + OUT_C); // Move Forward
    Wait(500);
    OnRev(OUT_C); // Turn Around
    Wait(200);
    OnFwd(OUT_C); // Come Back
    Wait(500);
    Off(OUT_A + OUT_C); // Stop
}

Move forward for 5 seconds, turn around, and come back
task main () {
    while (true) {
        moveForward(500);
        turnLeft(100);
    }
}

void moveForward(int time) {
    OnFwd(OUT_A + OUT_C);
    Wait(time);
    Off(OUT_A + OUT_C);
}

void turnLeft(int time) {
    OnFwd(OUT_C);
    OnRev(OUT_A);
    Wait(time);
    Off(OUT_A + OUT_C);
}

Drive around in a square shape
Using Sensors

- Recall that there are three sensor connections, and in NQC the sensors attached to them are called
  - SENSOR_1
  - SENSOR_2
  - SENSOR_3
- To use a sensor we must first define its type using `SetSensor`
  - `SetSensor(SENSOR_1, SENSOR_TOUCH);`
  - `SetSensor(SENSOR_2, SENSOR_LIGHT);`
- By default, touch sensors return 0 or 1 (1 if currently pressed)
- By default, light sensors return a value in the range 0..100, with 0 meaning no measured light
NQC Example 3: Sensor Polling

```c
#define SENSOR_2 2
#define SENSOR_TOUCH 3

int main() {
    // Set the sensors to be polled
    SetSensor(SENSOR_2, SENSOR_TOUCH);

    // Move forward until a bump is detected
    OnFwd(OUT_A + OUT_C);  // Move Forward

    // Move forward until a bump
    while (SENSOR_2 != 1)  // Until a Bump
        ;

    // Then stop
    Off(OUT_A + OUT_C);  // Then Stop
}
```

*Move forward until a bump*
NQC Example 4: Multi-Tasking

```
task main () {
    SetSensor(SENSOR_2, SENSOR_TOUCH);
    start patrol;
    while(SENSOR_2 != 1) {
        ;
    }
    stop patrol;
    Off(OUT_A + OUT_C);
}

task patrol () {
    OnFwd(OUT_A);
    while(true) {
        OnFwd(OUT_C);
        Wait(400);
        OnRev(OUT_C);
        Wait(200);
    }
}
```

Drive back and forth until a bump
NQC Links

- NQC was developed by Dave Baum and is now maintained by John Hansen
  - http://bricxcc.sourceforge.net/nqc/
- For the Macintosh the IDE of choice is MacNQC
- For Windows the IDE of choice is Bricx Command Center (bricxCC)
  - http://bricxcc.sourceforge.net/
  - BricxCC can also be used with BrickOS, LejOS, and pForth
BrickOS

- A open source project
- Offers the ability to program the RCX in C and in C++
- Improvements over environments which use the standard firmware include:
  - Native code execution (no interpretation!)
  - More memory available
  - Floating point available
  - Memory management, call stack
  - Fine grained hardware control, including raw mode IR
- In addition:
  - C and C++ as in gcc
  - Priority-based preemptive multitasking
  - Real process synchronization with POSIX semaphores
#include <config.h>
#include <dmotor.h>

int main(int argc, char *argv[]) {
    motor_a_speed(MAX_SPEED); motor_c_speed(MAX_SPEED);
    motor_a_dir(fwd); motor_c_dir(fwd); msleep(5000);
    motor_c_dir(rev); msleep(2000);
    motor_c_dir(fwd); msleep(5000);
    motor_a_dir(off); motor_c_dir(off);
    return 0;
}
BrickOS Example 2

```c
#include <config.h>
#include <dsensor.h>
#include <dmotor.h>

int main(int argc, char *argv[]) {
    motor_a_speed(MAX_SPEED); motor_c_speed(MAX_SPEED);
    motor_a_dir(fwd); motor_c_dir(fwd);
    while(SENSOR_2 > 0xf000)
    {
        ;
        motor_a_dir(off); motor_c_dir(off);
    return 0;
}
```

Move forward until a bump
BrickOS Links

- The BrickOS home page
RCXLisp

- RCXLisp is designed to allow a programmer to
  - Remotely control the RCX from a Common Lisp program running on a desktop computer
  - Write RCXLisp programs to run on the RCX
  - Create and compile RCXLisp programs for downloading to RCXs “on the fly”
  - Simultaneously control more than one RCX from a single MindStorms infrared tower
  - Set up a network of RCX units that can communicate with each other in a targeted manner (as opposed to the “broadcast manner” supported by LEGO’s kit).
The Two Parts of RCXLisp

- The term `RCXLisp’ actually refers to two related languages:
  - Remote RCXLisp: a collection of macros, variables, and functions for remotely controlling RCX units from a desktop machine.
  - RCXLisp Proper: a subset of Common Lisp that can be used to write programs for controlling RCXs (with LEGO’s firmware, or with Mnet extended firmware that supports wireless networking and most of the opcodes from LEGO’s firmware 1.0) from onboard the units.
Writing RCXLisp Code

- The following examples demonstrate RCXLisp basics:
  - On-board and remote control
  - Targeted communication
- These and other examples can be found in the RCXLisp documentation
(defconstant *MOVE-TIME* 100)
(defconstant *TURN-TIME* 85)

(deffunction (main :primary t) ()
  (set-effector-state (:A :C) :speed 7)
  (set-effector-state (:A :C) :power :on)
  (loop
    (set-effector-state (:A :C) :direction :forward)
    (sleep *MOVE-TIME*)
    (if (= (random 1) 0)
      (set-effector-state :C :direction :backward)
      (set-effector-state :A :direction :backward))
    (sleep *TURN-TIME*))
)

Have the robot go for a random walk
(Code running on the RCX)
(DEFUN play-all-sounds ()
  (with-open-com-port (p 1)
    (with-open-rcx-stream (s p :rcx-unit 12)
      (DOTIMES (num 6)
        (play-system-sound num s)
        (DOTIMES (x 10000)
          (* x 1))))))

Have the RCX with network number 12
play all 6 of its system sounds
(Running on the Desktop)
(DEFUN test ()
  (with-open-com-port (p 1)
    (with-open-rcx-stream (rcx1 p :rcx-unit 1)
      (with-open-rcx-stream (rcx2 p :rcx-unit 2)
        (full-speed-ahead2 rcx1 5 :backward)
        (full-speed-ahead2 rcx2 1 :forward))))
)

Part 1 - Tell RCX 1 to move backward at speed 5, then tell RCX 2 to move forward at speed 1
(Running on the Desktop)
RCXLisp Example 3: Multiple RCXs

(DEFUN full-speed-ahead2 (r s dir) ; r moves at speed s
  (LET ((result 0)) ; in direction dir
    (using-rcx r
      (set-effector-state '(:A :B :C) :power :off)
      (set-effector-state '(:A :C) :direction dir)
      (set-sensor-state 2 :type :touch :mode :boolean)
      (set-effector-state '(:A :C) :power :on)
      (LOOP
        (SETF result (sensor 2))
        (WHEN (AND (NUMBERP result)
          (= result 1))
          (RETURN)))
      (set-effector-state '(:A :C) :power :float))))

Part 2 - Tell an RCX to move at a speed in a direction until a bump sensor is pressed
(Running on the Desktop)
RCXLisp Links

- RCXLisp was developed by Frank Klassner
- The RCXLisp software can be found at
  - http://robotics.csc.villanova.edu
- The underlying Lisp environment is Xanalys LispWorks. A demo version (5 hour per session time limit) can be found at
Links for Some Other Programming Languages

- ADA
  - http://www.faginfamily.net/barry/Papers/AdaLetters.htm

- pForth