Grant Agreement number: 128959-CP-1-2006-1-GR-COMENIUS-C21 2006 – 2518 / 001 – 001 SO2

Project TERECoP

Study (Work package 2): A methodology for designing robotics-enhanced constructivist learning for secondary school students: appropriate technology-based environment

Technological issues: robotics

1. LEGO Mindstorms & its Environments

1.1. RCX & ROBOLAB

1.1.1. RCX Architecture

In 1949 Godtfred Kirk Christiansen made the first Lego blocks. In 1977 Lego created a line, at the moment known as TECHNIC, that incorporated gears, axes, bars, plates, universal joints … allowing to create models that revealed the engineering and the science of “how the practical things work”.

From year 1984, Lego started a strict collaboration with the MIT (Massachussets Institute of Technology) to integrate programming languages to the famous toy blocks. The common philosophy is the constructivist learning, that is the children learn better making things, so that they themselves develop their theories and obtain knowledge of their experiences.

The result of this collaboration is the MindStorms technology, a set of pieces that allow to design and to program with the appearance and behaviour that the child wishes. From 1998 it is available in two formats:

Robotic Invention System (RIS):

- Programmable brick RCX
- CD-ROM with RCX Code software to program RCX.
- 717 pieces of Lego (elements to mount transmissions and structures)
- 2 motors (9v)
- 2 concact sensors
- 1 light sensor
- several connection cables
- 1 infrared transmitting tower (connected to the host PC through a serial line or USB in the 2.0 version)

Lego MindStorms for schools, very similar to RIS but with educative aims::

- Programmable brick RCX
- CD-ROM with RCX Code software to program RCX.
- 717 pieces of Lego (elements to mount transmissions and structures)

- 2 motors (9v)
- 2 contact sensors
- 1 light sensor
- several connection cables
- 1 infrared transmitting tower (connected to the host PC through a serial line or USB in the 2.0 version)
- 1 activity guide for the teacher
- 4 construction models (car, insect, basket, house)

In the development of many projects the educative version of Lego Mindstorms has been used since it includes Robolab, one of the languages that are explained in the tutorial.

Programmable brick or RCX (Robotic Command Explorer) is a microcontroller able to control three inputs to which you can connect different sensors (contact, light, temperature…), three outputs to which you can connect different actuators (motors and lamps) and an infrared port for communications.

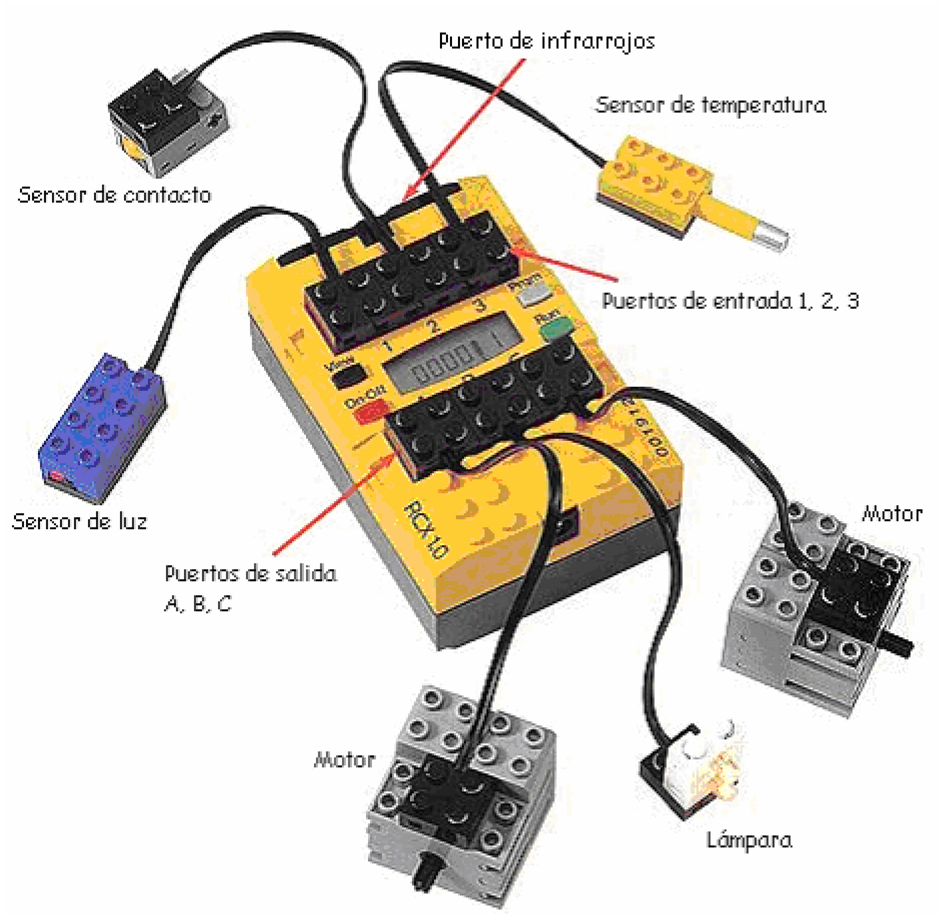


Figura 2.1. Programmable brick RCX with several actuators and sensors

The RCX assembles a microcontroller Hitachi H8/3292, with a H8/300 CPU Core which runs the control program. Through the device drivers of the H8/3292, the control program accesses to the devices of input/output of the RCX, such as buttons, loudspeaker and display (LCD). In addition, the sensors can be connected to the input ports of the RCX, providing the input of the sensors to the control program and this one can drive connected motor actuators through the output ports.

Ladrillo RCX

- Microprocesador Hitachi H8
- 16 KB ROM, 32 KB RAM
- 6 KB para programas de usuario
- Pantalla y cuatro botones
- Puerto de infrarrojos
- *Firmware* de LEGO
- Código interpretado
- Temporizadores
- Conversores A/D

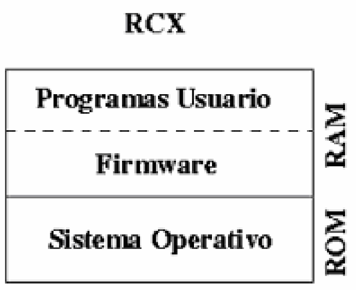
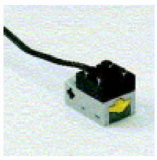
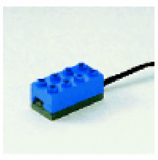


Figura 2.2. RCX chracteristics

Sensores

			
Sensor de contacto	Sensor de luz	Sensor de temperatura	Sensor de rotación

Actuadores

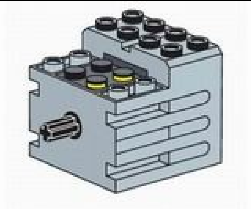
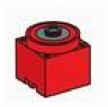
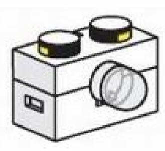
		
Motor 9v	Micromotor 9v	Lámpara 9v

Figura 2.3. The different actuators and sensors provided by Lego MindStorms

1.2.- NXT

1.2.1.- Introduction

The software that comes with the Lego Mindstorms NXT Edu kit is called NXT-G or extensively “Lego Mindstorms Education NXT” and it has a very effective and user friendly graphical interface. It is based on a large use of draggable icons for robotic commands (called “Blocks”). The GUI includes an area to set parameters for every command inserted in the user program, and an “educator” area with 39 complete demo examples (20 with the common palette and 19 with the complete palette). Every example is presented with a flash video (Challenge brief), a Building guide and a Programming guide.

Commands are collected in classes and divided into three palette: common and complete, for common and all commands respectively, and custom, to allow the definition of custom blocks. The development software includes most of the usual tools: file management (new, open, save), editing (cut, copy, paste), user profile (for personal blocks and programs). The help is an HTML/javascript based tool and therefore it can be navigated with a normal browser. The web support is facilitated through the easy access to the Mindstorms portal.

The software is powered by the following technologies: National Instruments LABView for the programming interface, Macromedia FLASH and Mozilla Gecko for the helping facilities.

1.2.2.- Programming

The user can open and rename one or more tabbed panels, each one representing a program which can be saved and retrieved. A program is saved into a file with a *.rbt* extension (called “Custom Pattern” type) with a proprietary format. Because the NXT protocol is based on very simple messages (a description is available in the documents regarding the Bluetooth Development Kit in http://mindstorms.lego.com/Overview/NXTreme.aspx), it appears that some kind of translation is done when a program is downloaded onto the NXT.

A program is formed by one or more *sequence beam*: any sequence represents a (sub)task which is executed concurrently on the NXT with the other tasks of the program. Sequence beams are linked one another through simple connectors (Fig. 1). The icon that represents a command block is decorated with numbers and graphical symbols to summarize the set of parameters for that command. Some blocks provide either optional or compulsory *data wires*: a data wire is used to carry information between blocks (e.g. a data wire is used to send out the random sample produced by a *Random* block). When allowed, data wires are set connecting pins showed by *data hubs* that can be displayed in the current block clicking the tab in the lower edge of its icon. Data wires allow a more dynamic control of the blocks.

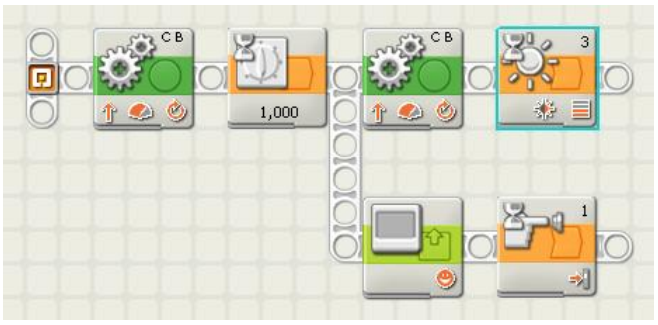


Fig. 1 – Sequence beams (source: Lego)

1.2.3.- NXT core, Motor and Sensors

<u>The NXT</u>
The technical details of the NXT brick are the following (fig. 1):

- 32-bit Atmel ARM7 (RISC) microcontroller
- 256 Kbytes FLASH, 64 Kbytes RAM
- Atmel 8-bit AVR microcontroller
- 4 Kbytes FLASH, 512 Byte RAM
- Bluetooth wireless communication (Bluetooth Class II V2.0 compliant)
- USB full speed port (12 Mbit/s)
- 4 input ports, 6-wire cable digital platform (One port includes a IEC 61158 Type 4/EN 50 170 compliant expansion port for future use)
- 3 output ports, 6-wire cable digital platform
- 100 x 64 pixel LCD graphical display
- Loudspeaker - 8 kHz sound quality. Sound channel with 8-bit resolution and 2-16 KHz sample rate.
- 4 buttons: Enter (orange), Menu arrows (2, light grey), Clear/Go back (dark grey)
- Power source: 6 AA batteries

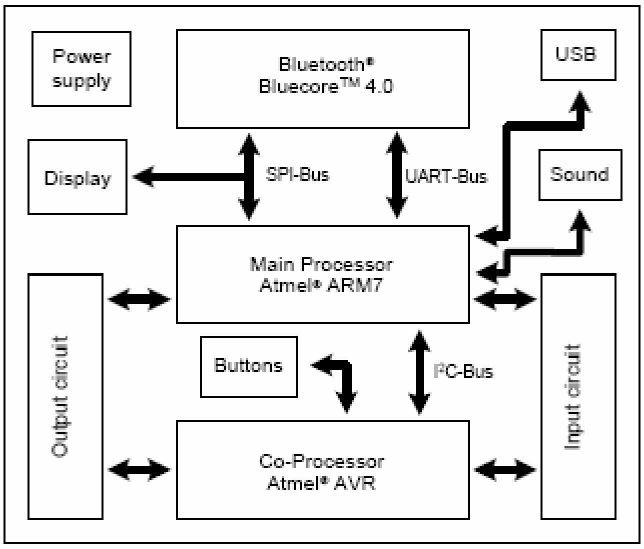
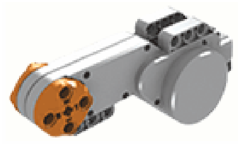
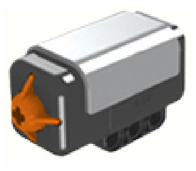


Fig. 2 - Hardware block diagram of the NXT Brick (Source: Lego)

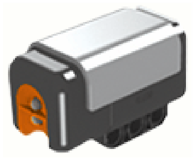


Motor
They are servo-motors with built-in rotation sensors with +/- 1 degre e of precision. This improvement (with respect RCX) permits a better control of the robot movement (see the move block below).

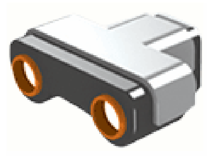


Touch sensor
Monostable switch.

Sound sensor
It can detect both decibels [dB] up to 90 dB and adjusted decibel [dBA, sensitivity calibrated to that of the human being]. The sound pressure is eventually reported in terms of percentage of the maximum level.



Light sensor
It reads the monochrome light intensity that is capture by its sensitiv e element.

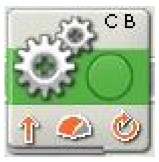


Ultrasonic sensor
It allows to detect objects and to measure distances from 0 and 255 cm with +/- 3 cm of precision. Two or more ultrasonic sensors operating in the same room may interrupt each other's readings.

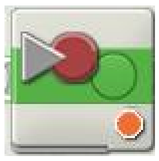
1.2.4.- Blocks

In this section the list of the available commands is presented. For every command a brief description of the function and of its parameters is provided.

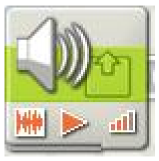
Common Palette



Move
It activates the outputs for a couple of motors.
Parameters: Selected port, Direction, Steering, Power, Duration, Next action
Data hub: yes



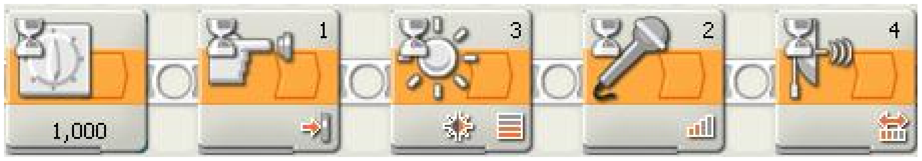
Record/Play
It records the sequence of actions of the robot and reproduces them.
Parameters: Action, Name, Recording, Time
Data hub: yes



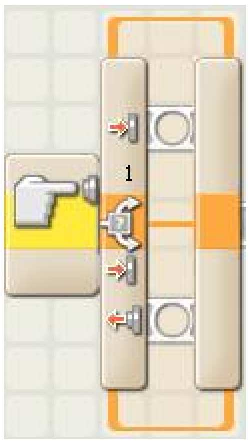
Sound
To produce a sound file or a tone.
Parameters: Action, Control, Volume, Repeat, [File][Note, Duration], Wait for Completion
Data hub yes

Display
To display something on the NXT's LCD screen.
Parameters: Action, Display clear, [File, Position], [Text, Position, Line], [Type, Position]
Data hub yes

Wait for
To wait for time or sensor.
Parameters: Control, [Seconds], [Sensor, {it depends on the chosen sensor}]
Data hub no



Loop
To repeat a subsequence.
Parameters: Control, Show counter, [{it depends on the chosen element}]
Data hub no

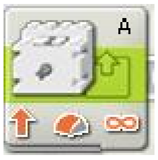


Switch
To chose a subsequence on the basis of some condition.
Parameters: Control, [Type, Sensor], Display Flat view,
[{it depends on the chosen element}]
Data hub no

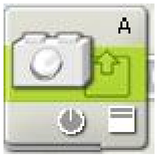
Complete Palette (apart from blocks in the common palette)

Motor
It activates the output for a single motor.
Parameters: Selected port, Direction, [Constant, Rump Up, Rump Down], Power, Duration, Wait for completion, Next action
Data hub: yes

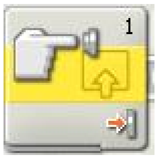
Send Message (through Bluetooth)
It sends a wireless message to another NXT.
Parameters: Connection number, [Text, Number, Logic], Destination mailbox number.
Data hub: yes



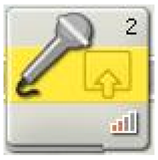
Motor*
It activates the output for a single motor of old style (RCX).
Parameters: Selected port, Direction, Power
Data hub: yes



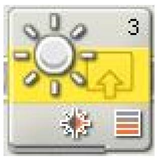
Lamp*
It turns on or off a lamp (RCX).
Parameters: Selected port, [On, Off], Intensity
Data hub: yes



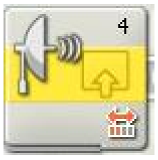
Touch sensor
It checks the sensor status at a specific point in the program, reporting a logical value.
Parameters: Port, [Pressed, Released, Bumped]
Data hub: yes



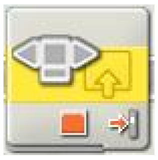
Sound sensor
It reports both a logical value (the sound is higher or lower a threshold) and a numerical value representing the sound level.
Parameters: Port, Level of comparison, [>, <]
Data hub: yes



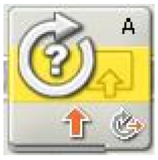
Light sensor
It reports both a logical value (the surrounding light is higher or lower a threshold) and a numerical value representing the light level.
Parameters: Port, Level of comparison, [>, <], Generate light
Data hub: yes



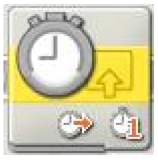
Ultrasonic sensor
It reports both a logical value (the distance is higher or lower a threshold) and the mea sured distance in inches or centimeters.
Parameters: Port, Level of comparison, [>, <], [Inches, Centimeters]
Data hub: yes



NXT buttons
It checks the status of one of the NXT buttons.
Parameters: [Enter, Left, Right], [Pressed, Released, Bumped]
Data hub: yes



Rotation sensor
It counts the number of degrees/full rotations of one motor e reports both a logical value (the counted number is higher or lower a threshold) and the counter value.
Parameters: Port, [Read, Reset], Direction, [>, <], Level of comparison, [Degrees, Rotations]
Data hub: yes



Timer
It reads one of the three built-in timer or resets it. If reading, it reports both a logical value (the timer value is higher or lower a threshold) and the timer value.
Parameters: [1,2,3], [Read, Reset], [>, <], Level of comparison

Data hub: yes

Receive message (through Bluetooth)

To receive a wireless message reporting both a logical value (the message contents is equal or not to a given value) and the message.
Parameters: [Text, Number, Logic], Compared value, [1..10]
Data hub: yes

Touch* sensor

It checks the sensor status at a specific point in the program, reporting a logical value (RCX).
Parameters: Port, [Pressed, Released, Bumped]
Data hub: yes

Rotation* sensor

It counts the number of ticks (16 ticks/rotation) e reports both a logical value (the counted number of ticks is higher or lower a threshold) and the counter value.
Parameters: Port, [Read, Reset], Direction, [>, <], Level of comparison
Data hub: yes

Light* sensor

It reports both a logical value (the surrounding light is higher or lower a threshold) and a numerical value representing the light level (RCX).
Parameters: Port, Level of comparison, [>, <]
Data hub: yes

Temperature* sensor

It reports both a logical value (the measured temperature is higher or lower a threshold) and a numerical value representing the temperature (RCX).
Parameters: Port, Level of comparison, [>, <], [Celsius, Fahrenheit]
Data hub: yes

Stop

It stops the program, motors and lamps.
Data hub: yes

Logic

It performs a logic function on its inputs or data wire inputs and gives out the result on an output data wire.
Parameters: [And, Or, Xor, Not], [Input, Data wired input]
Data hub: yes

Math

It performs a math function on its inputs or data wire inputs and gives out the result on an output data wire.
Parameters: [+, -, *, /], [Input, Data wired input]
Data hub: yes

Compare
For numerical comparison.
Parameters: [≤, =, ≥], [Input, Data wired input]
Data hub: yes

Range
It checks if a number is or not is within a range of numerical values.
Parameters: [Inside, Outside], [Input, Data wired input]
Data hub: yes

Random
Random number generator.
Parameters: Minimum, Maximum
Data hub: yes

Variable
It reads/writes a variable.
Parameters: [Logic, Number, Text], [Read, Write], [Value]
Data hub: yes

Text
Text concatenation.
Parameters: [Strings A/B/C, Data wired input]
Data hub: yes

Number to Text
Number to text conversion.
Parameters: [Number, Data wired input]
Data hub: yes

Keep alive
It will keep the NXT from entering sleep mode.

Data hub: yes

File access
File operations.
Parameters: [Read, Write, Close, Delete], Name, [Text, Number], Text
Data hub: yes

Calibrate
Sensor calibration.
Parameters: Port, [Light, Sound, Light*], [Calibrate, Delete], [Maximum, Minimum]
Data hub: yes

Reset motor
It resets the automatic error correction mechanism of the motors.
Parameters: Ports
Data hub: yes

Custom Palette

Myblock
It lists the customized blocks (selected blocks already put on the work area can be grouped into one new block).
Data hub: no

1.2.5.- Remarks

The NXT development software requires the adoption of an 'iconized' programming style which is rather efficient when the user has become confident with the interface and with the meaning of the most important blocks. Therefore, after a training period, which is shortened by the availability of the large library of examples, this approach is very effective for skilled users. Notwithstanding, in my opinion some critical aspect are worth to be pointed out.

- It is known that young students can find much more difficulty using an iconized interface instead of a textual programming language. The gap between the two approaches seems to have become even wider with respect to the previous version of Robolab for the enforced semantics of the new icons. Lucky many possibilities of interfacing NXT with programming languages at various levels are already available or under development.
- The choice to put Technic-style pieces in the NXT kit instead of the traditional brick-style of RCX seems to confirm the idea of the Lego developers to move the robotic segment to older users. This is furthermore confirmed by the possibility to control NXT with a complete LabView suite by advanced users.
- A general question must find answers during the experimentations (or possibly before): how problematic is the interference both among ultrasonic sensors and among bluetooth transmissions. This problem could condition the organization of the laboratory. One detail to be considered is that the software includes a tool to download the same program to multiple NXTs.
- NXT-G is a proprietary software whereas the NXT firmware is open-source and some useful documentation about the NXT hardware architecture is also available. This permits the interesting development of alternative solution to control the robot and the possibility to customize the NXT with auxiliary functions, both hardware and software.

2.- Other programming approaches for NXT

2.1.- Introduction

Besides NXT-G, Lego NXT may be controlled using different languages and environments that will be briefly described in this document. The alternative options examined in this document are the following:

- Robolab v. 2.9.x
- NXC/NBC/NPG
- RobotC
- Java/lejOS

- Myro
- Pyro
- Ruby/NXT
- URBI
- NXT-Symbian
- Microsoft Robotic Studio (MRS)
- JxLogo
- Microworlds
- MindSqualls .NET
- pbLua

Table 1 gives a comparison of relevant characteristics supported by some of the mentioned options.

2.2.- Analysis of the options

- **Robolab 2.9.x (http://130.64.87.22/robolabatceeo/)**

The actual version of Robolab supports NXT with specific icons and extensions. Such a version has been developed to support a smooth transition from RCX and NXT and it helps teachers to reutilize old projects in the new NXT context. Robolab will be completely replaced by the new NXT-G environment and its successors. Refers to the official documentation and suggested links below for any details.

Useful links:
http://ceeo.tufts.edu/
http://www.legoeducation.info/nxt/
http://thenxtstep.blogspot.com/
http://www.legoengineering.com/content/view/25/36/

- **NXC/NBC/NPG v. Beta 1.0.1 b27 (http://bricxcc.sourceforge.net/nbc/)**

Not eXactly C (NXC) is the descendant of NQC which was developed for RCX. It is a C-like high level language, built on top of NBC, usable for programming the NXT with easily understandable functional commands. The requested firmware on the NXT is the standard one. NBC is a simpler language like an assembly: it may be considered the assembly language of the virtual machine implemented by the interpreter running on the NXT. NPG is a very simple text language with a very limited number of commands with only implicit parameters: with NPG you can produce very quickly an elementary working program.

Just to have an idea of the level of programming in NXC consider the following example (from the NXC Tutorial) with straightforward function calls:

Features	NXT-G Retail	NXT-G Educational	RoboLab 2.9	NBC	NXC	RobotC	NI LabVIEW Toolkit	leJOS NXJ	pbLua
Language type	Graphic	Graphic	Graphic	Assembly	C-like	C	Graphic	Java	Lua
Firmware	Standard	Standard	Standard(#1)	Standard	Standard	Standard(#1)	Standard	Custom	Custom
IDE (included?)	Yes	Yes	Yes	Yes	Yes	Yes	No (#6)	No (Eclipse plugin coming)	No (#7)
Windows	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes (#7)
Mac OSX	Yes	Yes	Yes	Yes	Yes	Not Yet	Yes	Not Yet	Yes (#7)
Linux	No	No	No	Yes	Yes	No	No	Yes	Yes (#7)
Events	No	No	Yes	No	No	Yes	No	Standard Java events	
Multithreading	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	
Bluetooth Brick to PC	Yes	Yes	No	Yes	Yes	Yes	Yes	Not Yet	Not Yet
Bluetooth Brick to Brick	Yes	Yes	No	Yes	Yes	Not Yet	Yes	Not Yet	Not Yet
I2C Support	(#5)	(#5)	Yes	Yes	Yes	Yes	Yes	Not Yet	Not Yet
File System	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Not Yet	Not Yet
Floating Point	No	No	Yes	No	No	Yes	No?	Yes	(#8)
Datalog	No	No	Yes	No	No	Yes	No	No	No
How to get it	Included With retail version of NXT	Available with Educational Version of NXT	Available with Educational Version of NXT	BricxCC Web Site	BricxCC Web Site	CMU Web Site or LEGO Education	LabVIEW toolkit Site (#6)	Free download from lejos Web Site	pbLua Site
What do you want to do... (#2)									
Make robots move without learning to program	Yes	Yes	Yes						
Learn to Program using the NXT				No					
Write "Fast" programs				Yes	Yes	Yes		Maybe?	Yes
Write programs "Fast"	Yes	Yes	Yes						
Learn Advanced Programming concepts				?	Yes	Yes	Yes	Yes	Yes
Suited For (#3)	People starting with the NXT- or doing simple tasks	Schools starting with the NXT	Schools upgrading from the RCX to the NXT	Advanced programmer	C programmer	Applications requiring maximum speed	LabVIEW users, or people wanting to improve NXT-G	Java Programmers	Educational Users
	NXT-G Retail	**NXT-G Educational**	**RoboLab 2.9**	**NBC**	**NXC**	**RobotC**	**NI LabVIEW Toolkit**	**leJOS NXJ**	**pbLua**
Test Program (#4)									
Speed (loops/min)	720		73k		4285	93.9k	750/5350 **(#9)**		
Memory (bytes)	10704 bytes		559 bytes		1428 Bytes	561 bytes	8084/1890		
Time to write	10 minutes		~20 min		30 minutes	30 min	15 min/80 min		
Program	Code, Graphic		Code, Graphic		Code	Code1, Code2, Author's	Code, Graphic / Code1 +		

						comments	Code2, Graphic			
Other Comments (Facts & Opinions)	This software is designed for kids, but is not too limiting for adults	Same as Retail version, with different "Academy" robots.	LEGO has said this is the last version of RoboLab that will be made.				Can be used to create blocks that work in NXT-G programs, OR it can create programs to download directly to the NXT OR it can create PC programs to control the NXT(see below)		Early in development	

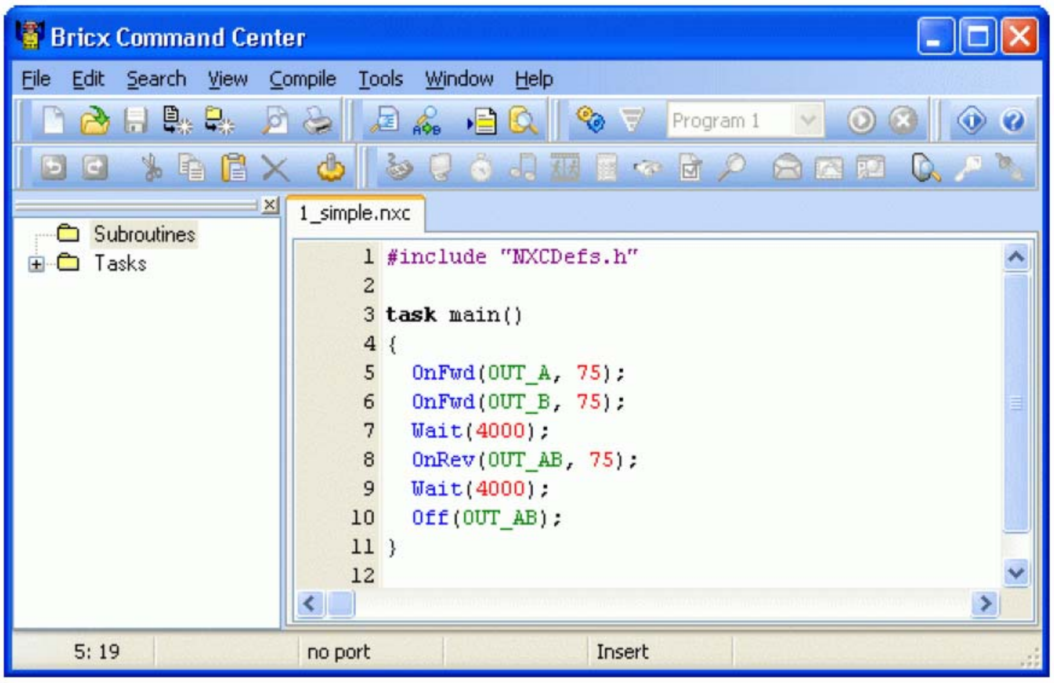
Table 1 – A comparison

```
#include "NXCDefs.h"

task main()
{
     OnFwd(OUT_A, 75);
     OnFwd(OUT_C, 75);
     Wait(4000);
     OnRev(OUT_AC, 75);
     Wait(4000);
     Off(OUT_AC);
}
```

A program is composed by one or more tasks, one of which is the *main* task; each task is formed by a sequence of commands which correspond, more or less, to the blocks of NXT-G. In the example, applied to a Tribot configuration, the two motors are separately activated at power 75. after 4 seconds, the movement reverted at the same power, and after other 4 seconds, it is stopped.

You can use any text editor to write the program and then compile it directly using the *nbc* command which accepts specific command line parameters. Alternatively you can use an IDE called *Bricx Command Center* (see fig. 1) that includes several useful features and mainly the possibility to easily compile and download the program onto the NXT.



```
#include "NXCDefs.h"

task main()
{
  OnFwd(OUT_A, 75);
  OnFwd(OUT_B, 75);
  Wait(4000);
  OnRev(OUT_AB, 75);
  Wait(4000);
  Off(OUT_AB);
}
```

Figure 1 – The *Bricx Command Center* main window

The NBC equivalent of the program above is the following:

```
#include "NXTDefs.h"

thread main
     OnFwd(OUT_A,75)
     OnFwd(OUT_C,75)
     wait 4000
     OnRev(OUT_AC,75)
     wait 4000
     Off(OUT_AC)
     exit
endt
```

In the next example in NXC, the program associates a light sensor with port 3, starts the movement and then, in an infinite loop, it makes the robot spin when the light value is bigger than the threshold and waits until the robot return to a darker position (e.g. if you wants the robot to follow a dark stripe on the floor).

```
#include "NXCDefs.h"
#define THRESHOLD 40

task main()
{
     SetSensorLight(IN_3);
     OnFwd(OUT_AC, 75);
     while (true)
     {
          if (Sensor(IN_3) > THRESHOLD)
          {
               OnRev(OUT_C, 75);
               Wait(100);
               until(Sensor(IN_3) <= THRESHOLD);
               OnFwd(OUT_AC, 75);
          }
     }
}
```

An equivalent in NBC is the following:

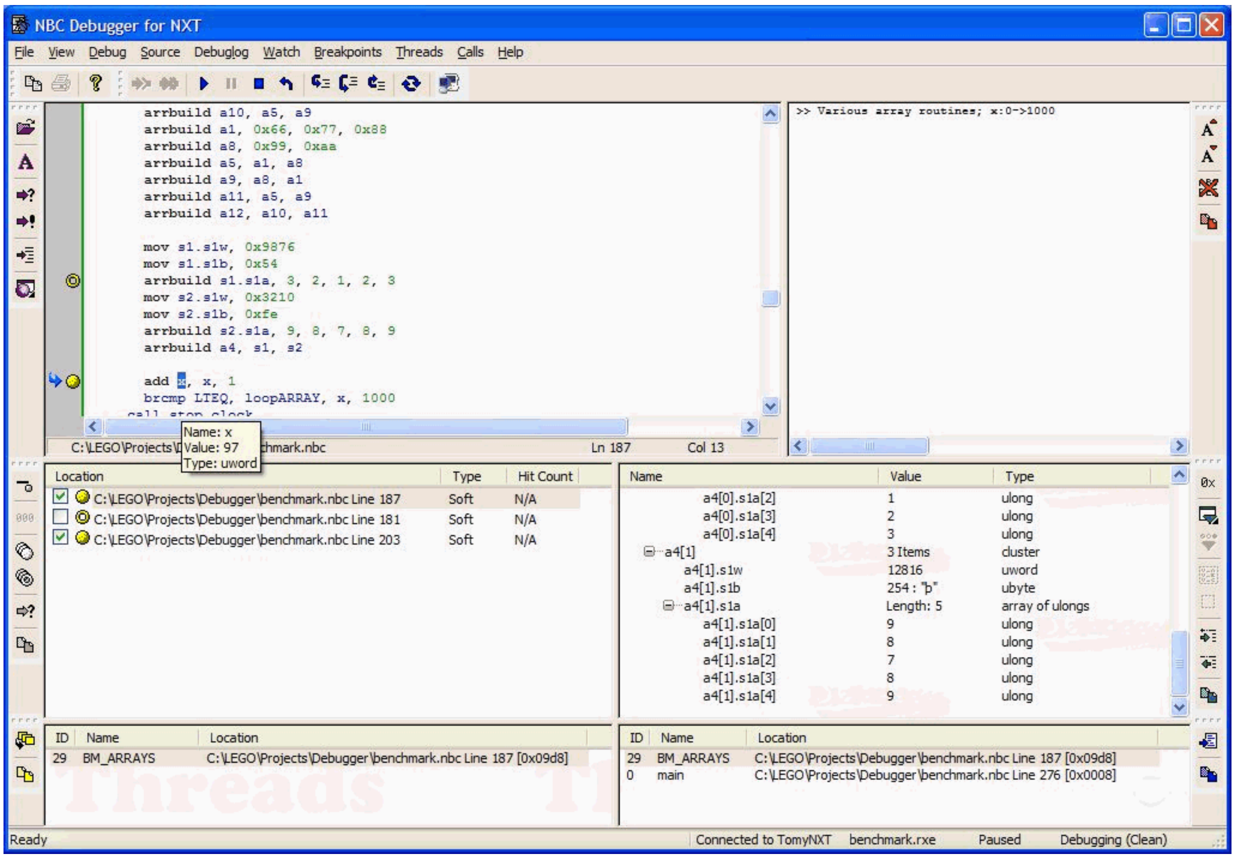
```
#include "NXTDefs.h"
#define THRESHOLD 40

dseg segment
     Level sword 0
dseg ends

thread main
     SetSensorLight(IN_3)
     OnFwd(OUT_AC,75)
CheckSensor:
     ReadSensor(IN_3,Level)
     brcmp LT, CheckSensor, Level, THRESHOLD
     OnRev(OUT_C,75)
FindLine:
     ReadSensor(IN_3,Level)
     brcmp GTEQ, FindLine, Level, THRESHOLD
     OnFwd(OUT_AC,75)
     jmp CheckSensor
endt
```

Macros and customized functions, like in the C/C++ language, gives the teacher a wide possibility to calibrate the level of commands to be used by students masking not appropriate commands, modifying the meaning of pre-defined commands, adding new very high level commands.

Another useful tool available is the **NBC debugger (http://www.sorosy.com/lego/nxtdbg/)** by which advanced users can remotely debug their applications, at three different level of deepness which correspond different percentage of increasing of the downloadable program size and speed. In fig. 2 you can see the main screen of the tool.


NBC Debugger for NXT
File View Debug Source Debuglog Watch Breakpoints Threads Calls Help

```
        arrbuild a10, a5, a9
        arrbuild a1, 0x66, 0x77, 0x88
        arrbuild a8, 0x99, 0xaa
        arrbuild a5, a1, a8
        arrbuild a9, a8, a1
        arrbuild a11, a5, a9
        arrbuild a12, a10, a11

        mov s1.s1w, 0x9876
        mov s1.s1b, 0x54
        arrbuild s1.s1a, 3, 2, 1, 2, 3
        mov s2.s1w, 0x3210
        mov s2.s1b, 0xfe
        arrbuild s2.s1a, 9, 8, 7, 8, 9
        arrbuild a4, s1, s2

        add x, x, 1
        brcmp LTEQ, loopARRAY, x, 1000
```

```
>> Various array routines; x:0->1000
```

```
Name: x
Value: 97
Type: uword
```

C:\LEGO\Projects\D…chmark.nbc Ln 187 Col 13

Location	Type	Hit Count
✓ C:\LEGO\Projects\Debugger\benchmark.nbc Line 187	Soft	N/A
C:\LEGO\Projects\Debugger\benchmark.nbc Line 181	Soft	N/A
✓ C:\LEGO\Projects\Debugger\benchmark.nbc Line 203	Soft	N/A

Name	Value	Type
a4[0].s1a[2]	1	ulong
a4[0].s1a[3]	2	ulong
a4[0].s1a[4]	3	ulong
⊟ a4[1]	3 Items	cluster
a4[1].s1w	12816	uword
a4[1].s1b	254 : "þ"	ubyte
⊟ a4[1].s1a	Length: 5	array of ulongs
a4[1].s1a[0]	9	ulong
a4[1].s1a[1]	8	ulong
a4[1].s1a[2]	7	ulong
a4[1].s1a[3]	8	ulong
a4[1].s1a[4]	9	ulong

ID	Name	Location
29	BM_ARRAYS	C:\LEGO\Projects\Debugger\benchmark.nbc Line 187 [0x09d8]

ID	Name	Location
29	BM_ARRAYS	C:\LEGO\Projects\Debugger\benchmark.nbc Line 187 [0x09d8]
0	main	C:\LEGO\Projects\Debugger\benchmark.nbc Line 276 [0x0008]

Ready   Connected to TomyNXT   benchmark.rxe   Paused   Debugging (Clean)

Figure 2 – The debugger main screen

Useful links:
http://robotica.irrepiemonte.it/robotica/linguaggi/doc/NBC_guida_ita_0_1x.pdf
 the translation of the tutorial in Italian includes a reference to an adaptation made by G.Marcianò in order to use the language with primary scholars.

- **RobotC (www.robotc.net)**

This is another environment around a C-like robotic language, recently developed at the Carnegie Mellon University, which support NXT. A commented example that uses a touch sensor is the following:

```
task main()
{
     while(SensorValue(touchSensor) == 0)
       //a while loop is declared with the touchsensor's
value
       // being 0 as it true condition
     {
          motor[motorA] = 100;
            //motor A is run at a 100 power level
          motor[motorB] = 100;
            //motor B is run at a 100 power level
     }
     motor[motorA] = -75;
```

```
        //motor A is run at a -75 power level
      motor[motorB] = -75;
        //motor B is run at a -75 power level
      wait1Msec(1000);
        //the program waits 1000 milliseconds before
        //running further code
}
```

- **Java/lejOS (http://lejos.sourceforge.net/)**

LejOS is a replacing firmware realizing a tiny Java Virtual Machine (from here ahead we call it NXT-JVM) in the NXT brick. This permits to write source programs in the Java language, preparing and compiling them with the standard tools, though with evident limitations with respect to the standard JVM. The NXT-JVM supports the following features:

- preemptive threads
- multi-dimensional arrays
- recursion
- synchronization
- exceptions
- most of Java basic types including floating numbers and String
- a well-documented robotic API.

The current version does not yet support all the sensors and advanced characteristics of the NXT architecture.

The BumperCar sample, extracted by the lejOS distribution, is presented with some comments. First two very simple *behaviours* are programmed, one representing ordinary motion (DriveForward) and one the reaction to an obstacle (HitWall).

```
// DriveForward.java
import lejos.robotics.*;
import lejos.nxt.*;

public class DriveForward implements Behavior {

   public boolean takeControl() { // dummy
      return true;
   }

   public void suppress() {  //  suppress  beh.  stopping
motors
      Motor.A.stop();
      Motor.C.stop();
   }

   public void action() {    // activate beh. switching on
motors

      Motor.A.forward();
      Motor.C.forward();
   }
}



// HitWall.java
import lejos.nxt.*;
import lejos.robotics.*;

public class HitWall implements Behavior {

   TouchSensor touch;

   public HitWall()
   {
        touch = new TouchSensor(Port.S2);
   }

   public boolean takeControl() {  // sensor’s status
      return touch.isPressed();
   }

   public void suppress() {   //   suppress   beh.   stopping
motors
      Motor.A.stop();
      Motor.C.stop();
   }

   public void action() {     // activate beh.
                              // reacting to the obstacle
      // Back up:
      Motor.A.backward();
      Motor.C.backward();
      try{Thread.sleep(1000);}catch(Exception e) {}
      // Rotate by causing only one wheel to stop:
      Motor.A.stop();
      try{Thread.sleep(300);}catch(Exception e) {}
      Motor.C.stop();
   }
}
```

The main program is represented by the *BumperCar* class which registers the two behaviours to be controlled by an *Arbitrator* whose responsibility is to regulate which behaviours should be activated on the basis of a sensor’s state (see the method `Behaviour.takeContrl()`). During an initial phase the *Arbitrator* class receives the array of behaviours to be scheduled.

```
// BumperCar.java
```

```
import lejos.robotics.*;
import lejos.nxt.*;

public class BumperCar {
   public static void main(String [] args) {
      Behavior  b1  =  new  DriveForward();    //  the  two
behaviours
      Behavior b2 = new HitWall();
      Behavior [] bArray = {b1, b2};
      Arbitrator arby = new Arbitrator(bArray);
        // set the arbitrator with the two behaviours
      Motor.A.setSpeed(200);      // set initial speed
      Motor.C.setSpeed(200);
      arby.start();  // activate the arbitrator
```

- **Myro (http://myro.roboteducation.org/)**

Myro is a new framework for programming robots, written in the interpreted language Python and designed for use in Introductory Computing courses. The interpreted language permits to interactively interface with the robot for very effective experimentations. This environment is being developed by the Institute for Personal Robots in Education (IPRE) of the Georgia Institute of Technology. Currently it supports only two robots, the Parallax's Scribbler and the Surveyor's SRV-1, but the supporting of NXT is promised.

- **Pyro (http://www.pyrorobotics.org/)**

Pyro stands for ***Py****thon* ***Ro****botics*. According to its web site: "the goal of the project is to provide a programming environment for easily exploring advanced topics in artificial intelligence and robotics without having to worry about the low-level details of the underlying hardware". Pyro is written in Python like Myro. It currently supports several robots (Pioneer, Hemisson, Sony AIBO, IntelliBrain-Bot, Roomba) and several simulation environment (Pyrobot, Stage, Gazebo, Robocup); the on-line documentation gives also some suggestions to interface specific personal robots. Pyro has the ability to define different styles of controllers, which are called the robot's brain. For example, the control system could be a neural network, behavior based, or a symbolic planner. One unique characteristic of Pyro is the ability to write controllers using robot abstractions that enable the same controller to operate robots with vastly different morphologies. And there is also a simulator which works with different robots and worlds. When this environment will support NXT, it will become a very interesting option for our project.

Useful links:
http://www.pyrorobotics.org/?page=PyroPublications
  publications on the subject
http://pyrorobotics.org/video/ch01-01.html
http://pyrorobotics.org/video/ch01-02.html
  two video demonstrations about the simulator (very interesting)

- **Ruby/NXT (http://rubyforge.org/projects/ruby-nxt/)**

Ruby is a object oriented interpreted programming language initially developed by Yukihiro "Matz" Matsumoto starting in 1993. Under the OO point of view, it is modern and 'pure' in the sense it has no primitive types (int, float, etc.) like Java and C++ but any run-time variable is an object (like in Smalltalk). It is also dynamic: new methods can be added to a class at runtime, at the cost of adding not so safe run-time type checking. It runs on the major OSs and includes most of the modern programming issues like Exceptions, IO support, Multithreading, and implements Iterators and other important design patterns. It was declared "Programming Language of the Year" in 2006 by TIOBE (http://www.tiobe.com/tpci.htm).

Ruby-NXT is a library that lets you control the NXT via Bluetooth using the Ruby language. There are three level of interfacing the robot: at low level, the NXTComm class provides direct access to the NXT Bluetooth bytecode protocol (there is also the UltrasonicComm class which implements the I2C communications needed to interact with the ultrasonic sensor, via NXTComm); at higher level, the NXT class provides multi-threaded, object-oriented interface to the motors, sensors, and most other core NXT functions, whereas the Commands module (included with NXTComm) provides a command object-based interface very similar to the Blocks in NXT-G.

Useful links:
http://ruby-nxt.rubyforge.org/

- **URBI (http://www.urbiforge.com/index.php)**

URBI (Universal Real-time Behavior Interface) is a simple but powerful way to control any robot or complex system like a video game, using a convenient and easy to use scripting language that can be interfaced with several popular programming languages (C++, Java, Matlab,...) and OS (Windows, Mac OSX, Linux). URBI is based on a client/server architecture, which give a great deal of flexibility. URBI includes powerful features compared to existing scripting solutions: parallel execution of commands, event programming, command tagging, dynamic variables,... Currently, URBI is used as well by academic research labs, the industry and by hobbyists.

Useful links:
http://www.gostai.com/lego.html

- **NXT Symbian (http://nxt-symbian.sourceforge.net/)**

It is just an application written in Java running on Symbian 6.0 Java-enabled mobile phones. It permits to send low-level commands to the NXT via the Bluetooth interface.

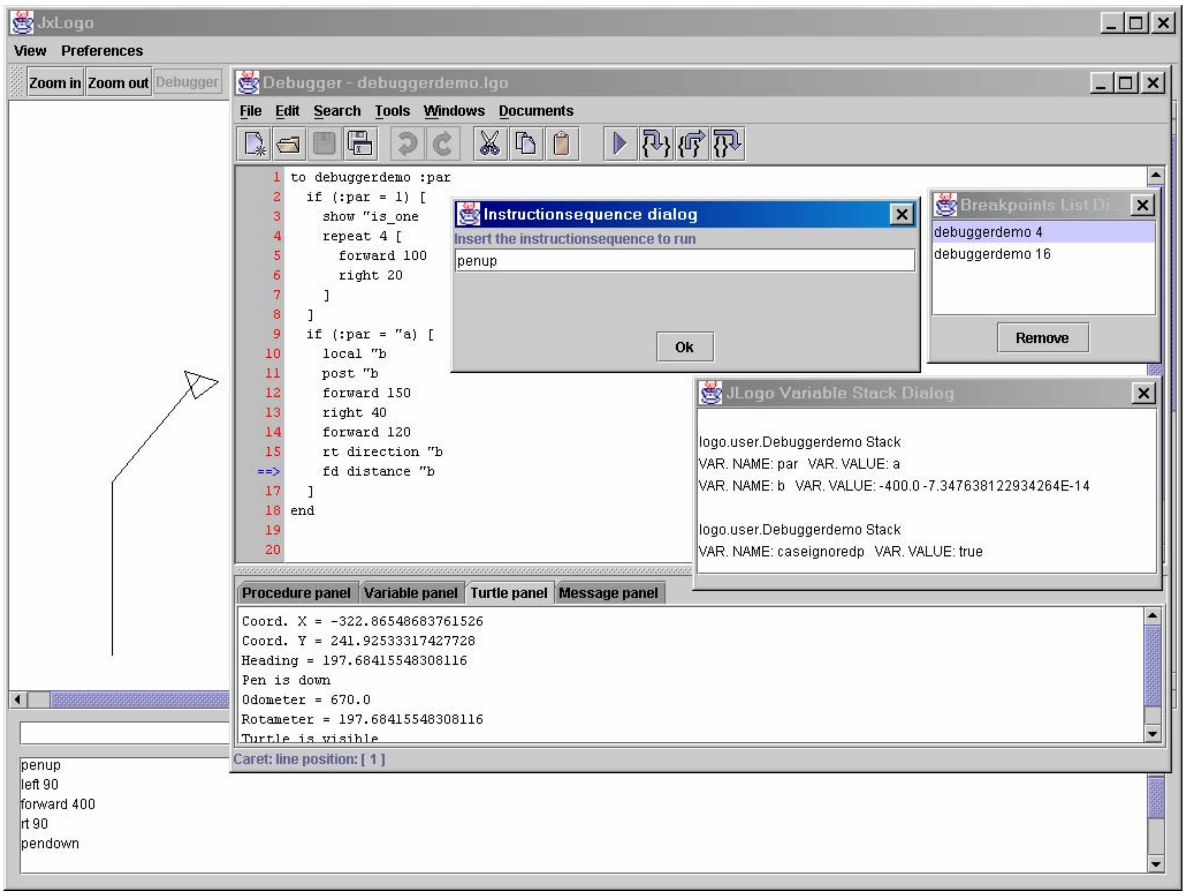
- **Microsoft Robotic Studio (MRS) (http://msdn.microsoft.com/robotics/)**

The following are some presenting words by the MRS Group: "Microsoft has created a new software development kit for the robotics community with the goal of supplying a software platform that can be used across a wide variety of hardware, applicable to a wide audience of users, and development of a wide variety of applications.". The environment includes:

- A scalable, extensible runtime architecture that can span a wide variety of hardware and devices. The programming interface can be used to address robots using 8-bit or 16-bit processors as well as 32-bit systems with multi-core processors and devices from simple touch sensors to laser distance finding devices.
- A set of useful tools that make programming and debugging robot applications scenarios easier. These include a high quality visual simulation environment that uses for software physics supplied by the Ageia Technologies PhysX engine.
- A set of useful technology libraries services samples to help developers get started with writing robot applications.

- **JxLogo (jxlogo.dei.unipd.it, currently under construction)**

JxLogo (fig. 3) is an undergraduate student project developed at Padua University and entirely realized in Java. Its goal is twofold: it is intended on the one hand, to encourage experimentations and usage of Java technologies by university students, on the other, to implement a complete and innovative environment to support advanced Logo programming. Starting from a strict compliance to MSWLogo for the name and meaning of the basic primitives, JxLogo includes the OO component as an actual extension of the traditional Logo language. The hoped progression in learning complex structures derives from the attention paid during the design of the language and partly from the interpreted nature of the language. Abstraction, visibility control, inheritance, polymorphism are all supported characteristics though through some simplifications reasonable for a programming environment developed for secondary school students and with general learning purposes.


JxLogo
View Preferences
Zoom in
Zoom out
Debugger
Debugger - debuggerdemo.lgo
File Edit Search Tools Windows Documents
Instructionsequence dialog
Insert the instructionsequence to run
Ok
Breakpoints List Di
Remove
JLogo Variable Stack Dialog
Procedure panel
Variable panel
Turtle panel
Message panel

```
 1  to debuggerdemo :par
 2    if (:par = 1) [
 3      show "is_one
 4      repeat 4 [
 5        forward 100
 6        right 20
 7      ]
 8    ]
 9    if (:par = "a) [
10      local "b
11      post "b
12      forward 150
13      right 40
14      forward 120
15      rt direction "b
==>     fd distance "b
17    ]
18  end
19
20
```

```
penup
```

```
debuggerdemo 4
debuggerdemo 16
```

```
logo.user.Debuggerdemo Stack
VAR. NAME: par   VAR. VALUE: a
VAR. NAME: b   VAR. VALUE: -400.0 -7.347638122934264E-14

logo.user.Debuggerdemo Stack
VAR. NAME: caseignoredp   VAR. VALUE: true
```

```
Coord. X = -322.86548683761526
Coord. Y = 241.92533317427728
Heading = 197.68415548308116
Pen is down
Odometer = 670.0
Rotameter = 197.68415548308116
Turtle is visible
```

Caret: line position: [1]

```
penup
left 90
forward 400
rt 90
pendown
```

Figure 3 – The JxLogo GUI

The integration of JxLogo with a robotic structure has been already realized through a basic RobotTurtle class which can communicate with Lego RCX (fig. 4). Its methods resembles the turtle commands but they are actually realized by the robot. For example you can interactively execute the following sequence:

```
make "RT RobotTurtle[] ; define a RobotTurtle instance
:RT'forward(15) ; apply commands to the instance
:RT'right(120)
:RT'forward(15)
:RT'right(120)
:RT'forward(15)
:RT'right(120)
```

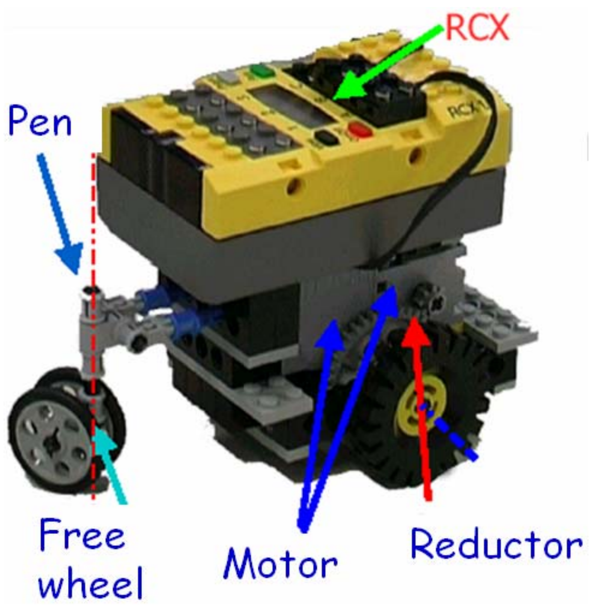


Figure 4 – The Robot Turtle

When a similar integration will be carried out with the NXT, both basic commands and complex actions will be programmable in a high level OO language but with the spirit of Logo. JxLogo may be easily extended to provide all the basic functions that are useful for our experimentations thanks to its known structure.

- **Microworlds EX Robotics Edition (http://www.microworlds.com/solutions/mwexrobotics.html)**

According to the web site this SW does not yet support NXT but we have rumours that this support is under development. MicroWorlds EX Robotics comes with all the features of MicroWorlds EX plus full programming capabilities for the Cricket robot, palm sized micro computer, and the LEGO RCX® Brick. Main features:

- Build a model and control it using the procedures you download to the Cricket or to the LEGO RCX® programmable brick.
- Test programming ideas by sending instructions from the computer via the LEGO® infrared tower to the RCX brick or via the Interface Cricket to the Cricket.
- Your model can also gather data as it runs - data that can be displayed using other MicroWorlds EX Robotics features.
- Use MicroWorlds' EX features to build an onscreen presentation or simulation in which an onscreen event triggers the model to start.

- **MindSqualls .NET (http://www.mindsqualls.net/)**

MindSqualls is a .Net 2.0 library for remote controlling NXT via a bluetooth connection. It is written in C# but it can be used with any of the .Net programming languages. It offers full support for all direct commands; it supports also the HiTechnic Compas sensor and, in the next version, the HiTechnic Color sensor. The example code on the site home page is the following:

```
// Create a NXT brick on COM40.
NxtBrick brick = new NxtBrick(40);
```

```
// Attach motors to port B and C on the NXT.
brick.MotorB = new NxtMotor();
brick.MotorC = new NxtMotor();

// Connect to the NXT.
brick.Connect();

// Run them at 75% power, for a 3600 degree run.
brick.MotorB.Run(75, 3600);
brick.MotorC.Run(75, 3600);

// Disconnect from the NXT.
brick.Disconnect();
```

- **pbLua (http://www.hempeldesigngroup.com/lego/pbLua)**

PbLua derives from Lua (http://www.lua.org/), a powerful light-weight programming language designed for extending applications. Lua is also frequently used as a general-purpose, stand-alone language. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics.
PbLua is a text-based language for the NXT and exploits the extendible features of Lua:

- It's written in portable C, with minimal runtime requirements
- It can be compiled on the fly on the target machine, which is the NXT in our application
- It's a small, easy to read, and easy to write language
- It has extensive documentation available online, and a very friendly newsgroup

The following line program presented in the site home page follows.

```
-- All NXT api functions are in the nxt table, so
nxt.apiname() is the
  -- standard way to access those functions

  function FollowLine()
    -- Set up sensor 3 to be a light sensor
    nxt.InputSetDigi0(3)
    nxt.InputSetDirOutDigi0(3)

    -- Motors 1 and 2 are in Brake Mode and have regulation
enabled
    nxt.OutputSetMode(1,2)
    nxt.OutputSetMode(2,2)

    nxt.OutputEnableRegulation(1,1)
    nxt.OutputEnableRegulation(2,1)
    -- Set up the light and dark thresholds and the motor
speeds
    local T1 = 600
    local T2 = 630
    local SpeedSlow = 50
    local SpeedFast = 100

    -- LoopCount is just to see how fast Lua is running the
line
    -- following loop for 10 seconds
    local LoopCount = 0

    -- Read the msec timer and get ready to run the loop
for 10 seconds
    local t=nxt.TimerRead()+10000

    while t > nxt.TimerRead() do

      -- Read the light sensor and save the result
      local SV = nxt.InputGetRawAd(3)

      -- If the sensor is reading below the threshold
(white) turn towards
      -- the line
      if SV < T2 then
        nxt.OutputSetSpeed(1,32,SpeedFast,0)
      else
        nxt.OutputSetSpeed(1,32,SpeedSlow,0)
      end

      -- If the sensor is reading above the threshold
(black) turn away
      -- from the line
      if SV > T1 then
        nxt.OutputSetSpeed(2,32,SpeedFast,0)
      else
        nxt.OutputSetSpeed(2,32,SpeedSlow,0)
      end

      -- Count the number of times we've run the loop
      LoopCount = LoopCount + 1

    end

  -- Make room on the LCD for a line of text, and print the
number
  -- of times we ran through the loop
  nxt.DisplayScroll()
  nxt.DisplayText(string.format("%i",LoopCount))

  -- Don't forget to turn the motors off!
  nxt.OutputSetSpeed(1,0,0,0)
  nxt.OutputSetSpeed(2,32,0,0)

end

-- Now run the FollowLine function we just defined!
FollowLine()
```

2.3.- Remarks

The wide range of alternatives to command the NXT is not surprising. The NXT virtual machine is rather simple and can be easily interfaced with a controlling environment through the well documented API and protocol. Hence we can even expect that the number of possibilities grows in the next future. For the aims of our project it is particularly interesting, on the one hand to exploit different approaches, some using iconic graphical interfaces, some others based on more or less specialized programming languages; on the other hand, the possibility to adapt or even to create an environment with specific commands and constructs that will be possibly claimed as result of the methodological analysis.
Provided that a graphical language is more suitable for advanced users, together with NXT-G the new MRS environment must be analyzed in order to check if and in which occasions it can substitute the NXT-G 'native' approach. MRS seems a promising system to uniformly support different robotic architectures and makes it available very interesting tools. The other textual language-based approaches are advisable in case of younger users. Among them, considering JxLogo, I have already offered to the TERECoP community to adapt its future developments to the requests of the project. In my opinion Pyro and Ruby-NXT are very interesting options due to the specific characteristics of their host language (Python and Ruby respectively).

3. Some other robotic architectures

3.1.- Cricket

The Handy Cricket is based on the microcontroller PIC16C715, who includes 2048 bytes of ROM once programmable ROM (burned with the system of operation of the Handy Cricket), analogous, digital inputs and outputs. The PIC is the "brain" of the Handy Cricket. Through PIC, the control program accesses to the I/O devices of the Handy Cricket, such as buttons and piezobeep. In addition, the sensors can be connected to the input ports of the Cricket, allowing the sensors to condition the control program and this can drive motor actuators connected to the output ports.

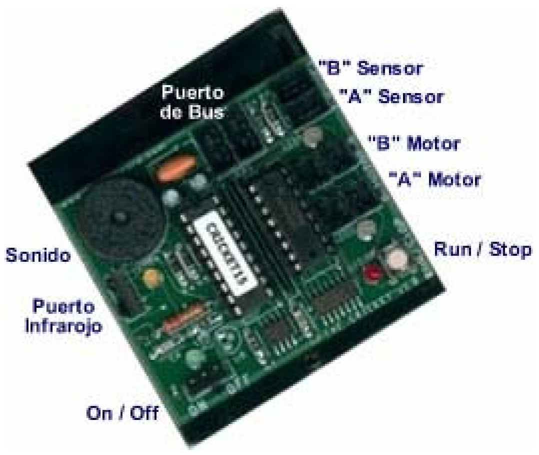


Figura XXX: El handy Cricket(http://www.gleasonresearch.com/)

The Cricket has the following input devices:

- 1 On/Off switch.
- 1 Run/Stop button.
- Input ports A and B.
- Voltage level meter
- Counter
- Infrared receiver

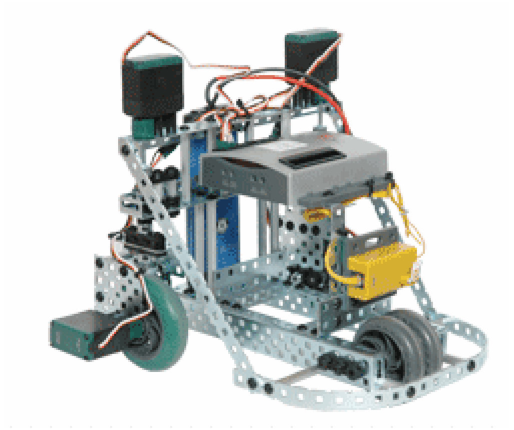
and the following output devices:

- Piezobeep
- 2 ooutput ports A e B.
- Each port has associated a two-color LED pencil (red-green) that indicates the sense of rotation of the motor
- Infrared trnsmitter.
- In addition two expansion boards type BUS.

2.2.- Handyboard

The Handy Board (http://www.handyboard.com/) is a 6811-based microcontroller system that lets you build mobile robots for educational, hobbyist, and industrial purposes. People use the Handy Board to run robot design courses and competitions at the university and high school level, build robots for fun, and control industrial devices. This site is the home page and resource center for users of the Handy Board.

2.3.- Vex Robotics Design System



The Vex Robotics (http://www.vexlabs.com/) Design System (Starter Kit) contains everything you need to design a robot. The creative possibilities are endless. The Starter Kit was inspired by the FIRST (For Inspiration & Recognition of Science & Technology) Robotics Competition--the world's leading high school robotics program. This not-for-profit org. was founded to inspire young people's interest and participation in science and technology.