LEGOSC
Mindstorms NXT robotics programming for artists

Jorge Cardoso
Research Centre for Science and Technology in Art (CITAR), Portuguese Catholic University, Rua Diogo Botelho 1327, 4169-005 Porto, Portugal
jccardoso@porto.ucp.pt

Manuel Ferreira, Cristina Santos
Department of Industrial Electronics, University of Minho, Campus de Azurem – Guimarães, Portugal
mjf@dei.uminho.pt, cristina@dei.uminho.pt

Keywords: Robots, Art, Lego Mindstorms, OSC.

Abstract: Robotics is an interesting but difficult area for digital artists who generally don't have much academic background on electronics or computer programming. Digital art students normally use high-level application to program their visual and sonorous installations. This paper presents LegOSC - a tool that allows the control of the Mindstorms NXT robots from any application that uses the Open Sound Control protocol which is implemented by most of those high-level applications. This allows artists to create works which incorporate robotic parts using the familiar programming environment.

1 INTRODUCTION

Robotics are becoming increasingly interesting for artists in many areas, e.g., painting (Moura and Pereira, 2004), theater (Ullanta, 2007), sculpture (Pisaturo, 2007), installation (da Costa, 2007), music (f18institute, 2007). More and more, art work incorporates some electro-mechanic parts which provide more ways for the artist to express himself, or to complement his ability to do so.

However, using robotic systems still requires some expertise that most artists don't possess. Even in digital art degrees, robotics is usually not a subject. Art students generally lack the necessary background in electronics.

Altough there are now some simple tools to build and program robotic systems (of which, perhaps, the most widely known is the Lego Mindstorms (Lego Group, 2007)) and many uses in classrooms (Fagin, 2003; Klassner and Anderson, 2003; Bruder and Wedeward, 2003; Ceccarelli, 2003), these can still be difficult to integrate in an art work. As an example, students in the author's school usually use platforms like Processing (Fry and Reas, 2007), Eyesweb (Camurri et al., 2000), Adobe Flash (Adobe, 2007b), Adobe Director (Adobe, 2007a), Max/MSP (Cycling74, 2007), Pure Data (Puckette, 1996), to implement their visual and sonorous installations. These platforms can (and usually are) be interconnected using MIDI messages (`http://www.midi.org`), or Open Sound Control (OSC) (Wright and Freed, 1997) messages.

In order to provide an easier setting for the use of a robotic system that can be controlled by a platform like the ones listed above, LegOSC has been implemented – an Open Sound Control gateway application to control the Lego Mindstorms NXT robotic system.

The rest of this paper is organized as follows: section 2 introduces the Lego Mindstorms NXT system; section 3 describes the Open Sound Control protocol; section 4 presents the arquitecture and usage of the LegOSC application; section 5 describes some usages of LegOSC; section 6 describes some of the limitations of LegOSC; finnaly, section 7 concludes.

2 MINDSTORMS NXT

The Lego Mindstorms NXT system consists of three main component types: the NXT brick; motors and sensors and assorted Lego bricks.

The NXT brick has a 32-bit ARM7 microcontroller, 256 Kbytes FLASH, 64 Kbytes RAM. It has Bluetooth wireless communication (Bluetooth Class II V2.0 compliant) a USB full speed port (12 Mbit/s),

4 input ports, 3 output ports, 100 x 64 pixel LCD graphical display and a loudspeaker - 8 kHz sound quality. Figure 1 shows the block diagram of the NXT brick.

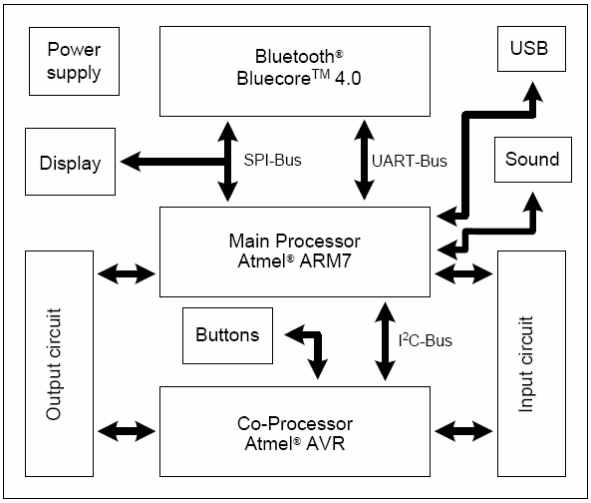

Figure 1: Hardware block diagram for the NXT brick, from (Lego Group, 2006b).

Three servo-motors can be connected to the three output ports and sensors (light, ultrasonic, pressure, sound, etc) to the four input ports.

The rest of the Lego bricks allow the construction of various shaped and sized robots.

The Mindstorms NXT robots can be controlled by uploading a program to the NXT brick and have it run in an autonomous fashion. These programs can be written using the Mindstorms NXT visual programming software. In alternative, one can use other languages with sintaxes close to C, such as "Not eXactly C" (`http://bricxcc.sourceforge.net/nbc`) or with Java sintaxes such as "LeJos" (`http://lejos.sourceforge.net`), although some may require changing the NXT firmware.

The robot can also be controlled wirelessly by using the Bluetooth Direct Commands protocol (Lego Group, 2006a). This protocol allows sending instructions to actuate the motors or read sensors without the need to previously upload a program to the NXT brick. It also provides a way to interface the robot with other programs that understand this bluetooth protocol.

2.1 Bluetooth Protocol

Figure 2 shows the block diagram for the communication between a PC and Lego NXT.

Communication can be accomplished by using an

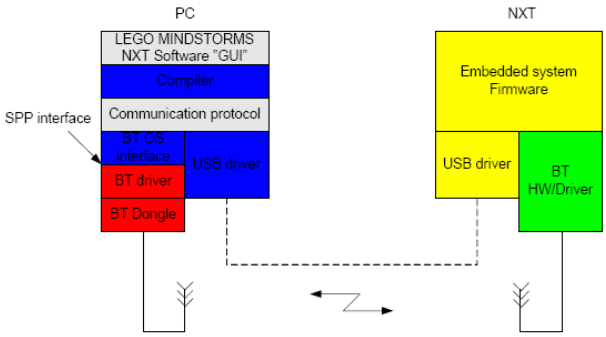

Figure 2: Communication block diagram, from (Lego Group, 2006a).

USB cable connecting the PC and the NXT or wirelessly by using bluetooth.

The protocol can be used to (based on (Lego Group, 2006a)):

1. Read, write and delete files.
2. Direct communication with the NXT system to:
   - Send direct commands to the virtual machine.
   - Send message commands to program mailboxes.
   - Get file list within NXT.

The bluetooth protocol package is shown on Figure 3.

Bytes 0 and 1 are the LSB and MSB bytes, respectively, of the length of the command data.

Byte 2 is the command type. The 7 least significant bits identify the command type and the most significant bit (bit 7) determines if the command requires a reply from the NXT, or not. The command type can by one of the following:

- `0x00`: Direct command, reply required.
- `0x01`: System command, reply required.
- `0x02`: Reply command.
- `0x80`: Direct command, reply not required.
- `0x81`: System command, reply not required.

Byte 3 identifies the command.

Byte 4-N are the command specific data.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4		Byte N
Length, LSB	Length, MSB	Command Type	Command			

Figure 3: Bluetooth Protocol package.

LegOSC uses only the Direct Commands subprotocol to communicate with the NXT.

3 OPEN SOUND CONTROL

The Open Sound Control is an application level communication protocol. It was meant to replace MIDI but, although it was not successfull at that, it has become a widely used protocol in sound synthesis and video processing applications and many general purpose programming environments (see `http://www.cnmat.berkeley.edu/OpenSoundControl/` for a more comprehensive list).

OSC is a simple message based, transport-independent protocol, although most of its implementations use UDP or TCP as the transport layer. OSC messages have an address and a variable number of typed arguments. OSC standard types include 32-bit integers and floats, strings, blobs and 64-bit fixed point timetags.

An OSC Message consists of the following parts:

Address Pattern	Type Tag	Arg 0	...	Arg n

The OSC Address Pattern is an OSC String[1] that starts with the '/' character. The OSC Address Pattern is pattern-matched by the receivers to decide if a message should be delivered.

The Type Tag is also an OSC String in which each character represents the type of an OSC Argument in the message.

Each OSC Message may have a variable number of binary represented arguments. Each argument representation is padded with zeroes to make it a multiple of 4.

4 LEGOSC

LegOSC is a gateway application that translates a set of pre-defined OSC messages into Bluetooth Direct Commands for the NXT brick, and vice-versa, as shown in Figure 4.

To configure LegOSC we need to define the local UDP port on which it will listen for OSC messages, the IP address and port of the OSC Application that will be communicating with LegOSC (and listening for OSC messages) and the virtual COM port on which the NXT Brick was connected.

Figure 5 shows a screenshot of the LegOSC application.

LegOSC will listen for OSC messages that tell it to actuate the motors or to read sensor values. In case of the latter, it will respond with another OSC message with the sensor value. For some applications however,

[1] An OSC String is null-terminated string of ASCII characteres, padded with nulls to make the total number of characters a multiple of 4.

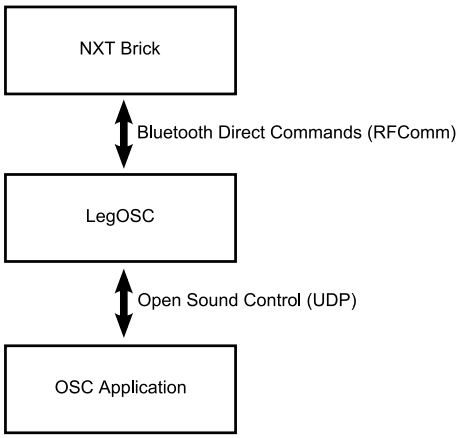

Figure 4: Communication between the NXT brick, LegOSC and the OSC Application.

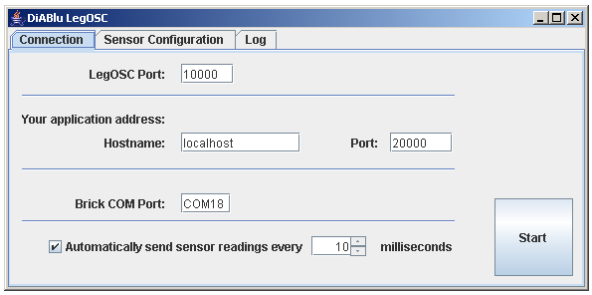

Figure 5: LegOSC Application.

reading sensors will be a continous operation so, in order to save some OSC traffic, LegOSC can be programmed to continously send sensor values without beeing asked for. The user can tell LegOSC that automatic readings are required and how often a reading should be made.

In order to be able to read a sensor (automatically), LegOSC must know the type of sensor connected to each port in the NXT brick. The user can tell LegOSC the type of sensor using the Sensor Configuration tab (Figure 6).

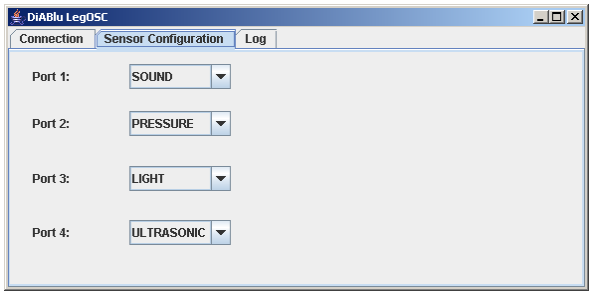

Figure 6: Configuring automatic sensor reading.

4.1 OSC Messages

The list of currently implemented OSC messages that LegOSC understands is:

- /motorForward ii – The first integer argument is the motor number and the second is the power to apply to the motor. This message will make the specified motor start to rotate at the specified power.
- /motorSlowStop i – The integer argument is the motor number. This message stops the specified motor without aplying "brakes".
- /motorBrake i – The integer argument is the motor number. This message stops the specified motor and aplies "brakes".
- /resetMotor i – The integer argument is the motor number. This message resets the tachometer of the specified motor.
- /getMotorTachoCount i – The integer argument is the motor number. This message asks for the current tacho count of the specified motor and originates a /motorTachoCount message as the reply.
- /getButtonState i – The integer argument is the port to which the pressure sensor is attached. This message asks for the current state of the pressure sensor and originates a /buttonState message as the reply.
- /getLightLevel i – The integer argument is the port to which the light sensor is attached. This message asks for the current value of the light sensor and originates a /lightLevel message as the reply.
- /getSoundLevel i – The integer argument is the port to which the sound sensor is attached. This message asks for the current value of the sound sensor and originates a /soundLevel message as the reply.
- /getProximityLevel i – The integer argument is the port to which the ultrasonic sensor is attached. This message asks for the current value of the ultrasonic sensor and originates a /proximityLevel message as the reply.
- /getBatteryLevel – This message asks for the current voltage of the battery of the NXT.

Some of the above messages generate a response:

- /motorTachoCount ii – Response to /getMotorTachoCount. The first integer argument is the motor number; the second integer argument is the current tacho count for that motor.
- /buttonState ii – Response to /getButtonState. The first integer argument is the port number to which the pressure sensor is attached; the second integer argument if the current state of the pressure sensor (0 – not pressed; 1 – pressed).
- /lightLevel ii – Response to /getLightLevel. The first integer argument is the port number to which the light sensor is attached; the second integer argument if the current value of light sensor.
- /soundLevel ii – Response to /getSoundLevel. The first integer argument is the port number to which the sound sensor is attached; the second integer argument if the current value of sound sensor.
- /proximityLevel ii – Response to /getProximityLevel. The first integer argument is the port number to which the ultrasonic sensor is attached; the second integer argument if the current value of ultrasonic sensor.
- /batteryLevel i – Response to /getBatteryLevel. The integer argument is the current voltage in millivolts.

4.2 Java Libraries

LegOSC was written using the Java programming language and as a by-product of developing this gateway we developed a Java library that implements the NXT Bluetooth low-level commands as well as a higher level library that abstracts these low-level commands into higher level NXT-related objects. The class diagram is shown on Figure 7 (details of each class are hidden to save space).

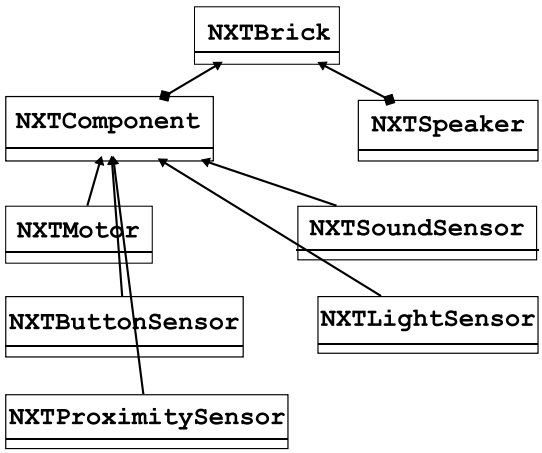

Figure 7: High-level Java library class diagram.

Since we had to develop this Java library and since the modifications were small, we decided to adapt the Java library to a Processing library. Processing

is a tool/programming languaged widely used in digital arts, so it made sense to allow direct control of the NXT without the need to use a different program. Processing libraries are usually composed of a single class to simplify its usage as much as possible. The (single) class diagram for this library is shown on Figure 8.

LegoNXT
Operations
public LegoNXT(PApplet parent, String commPort)
public boolean playTone(int frequency, int duration)
public boolean motorForward(int motorNumber, int power)
public boolean motorForwardLimit(int motorNumber, int power, int tachoLimit)
public boolean motorHandBrake(int motorNumber)
public boolean motorStop(int motorNumber)
public long getMotorTachoLimit(int motorNumber)
public long getMotorTachoCount(int motorNumber)
public long getMotorBlockTachoCount(int motorNumber)
public long getMotorRotationCount(int motorNumber)
public void resetMotorPosition(int motorNumber, boolean relative)
public boolean getButtonState(int portNumber)
public int getDB(int portNumber)
public int getLight(int portNumber)
public void setLightSensorType(int type, int portNumber)
public int getDistance(int portNumber)
public int getBatteryLevel( )
public void dispose( )

Figure 8: Processing library class diagram.

5 Example usages

To give a better idea on how LegOSC can be used, a brief description of some typical exercises and implementations that students are asked to do is given next.

The first one is a non-classic way of controlling a robot. Instead of using a mouse/joystick/keyboard, students are asked to think of a way to control a robot using sound. Figure 9 shows an implementation using Pure Data – an audio synthesis platform. In this example, a stereo microphone is used to control two motors. The sound level at each channel drives its own motor, thus enabling the user to direct the robot by making sound lauder at one microphone or the other.

The second example deals with mapping some physical parameter of the robot, e.g., distance to a wall, light level reading, into another type of signal. In this example (Figure 10) a simple Theremin was implemented. Readings from the proximity sensor and from the light sensor are used to drive two audio oscillators that together generate a frequency modulated audio signal. In this case the movement of

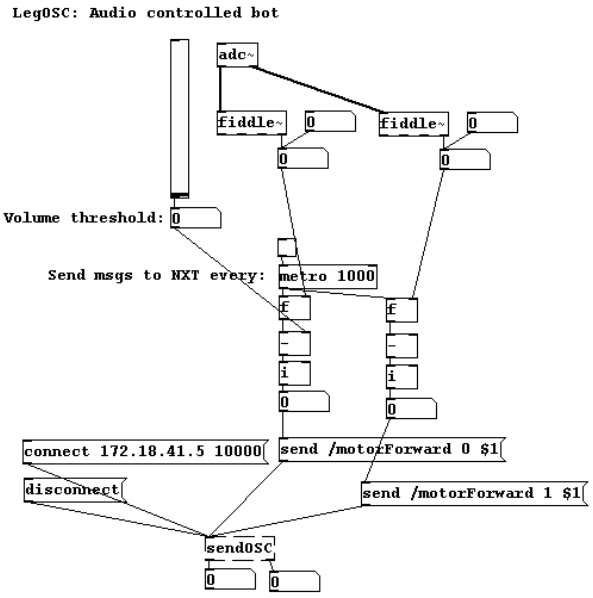

Figure 9: Pure Data implementation of a sound controlled robot.

the robot (which could be programmed in a diferente number of ways) generates an audio signal.

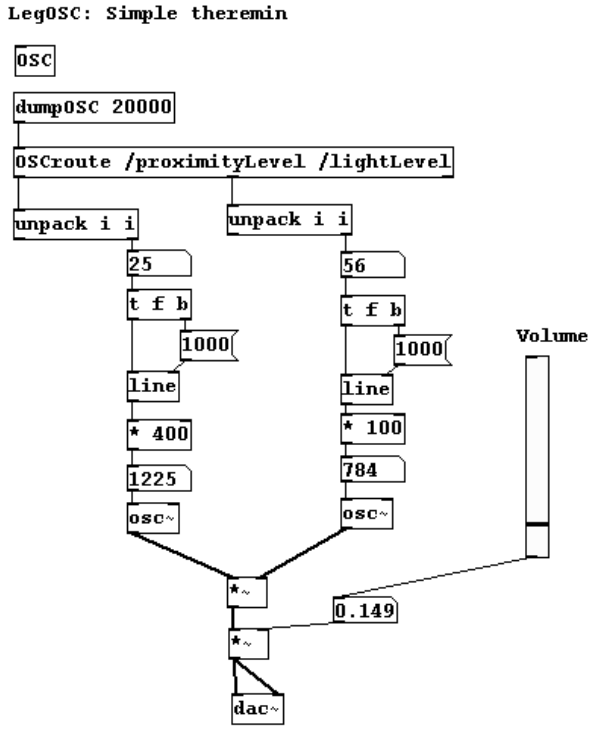

Figure 10: Pure Data implementation of a robot theremin.

Both examples were implemented in Pure Data

and the figures show the complete program. Pure Data is and audio synthesis and manipulation platform so it simplifies the kind of programming needed to implement the examples.

6 LIMITATIONS

LegOSC was developed with an education and artistic purposes in mind. It is not intended for precision robotics.

The understood messages were kept as simple as possible in order to allow basic control of the robots but not to overwhelm the student/artist with detail.

The bluetooth communication latency may also make it unsuitable for some applications where a rapid response to an event is required.

7 CONCLUSION AND FUTURE WORK

We have developed an Open Sound Control gateway application for controlling the Lego Mindstorms NXT robotics system. This application is intended to be used by digital art students as a simple way to control and integrate robotic art work with other often used systems to develop visual and sonorous installations.

We hope it will allow an easier first approach to teaching and using robotics in the digital arts area.

LegOSC is currently being used by the author's students and we hope to enhance it with the experience gained with its usage.

One of the improvements that we have already gathered from experiences is the ability to control several NXT using the same LegOSC instance. This will facilitate the programming of applications that make use of several robots at the same time.

The tool is freely available for download at `http://diablu.googlecode.com/svn/trunk/LegOSC/`. The Processing library is also available from the Processing site at `http://processing.org`.

REFERENCES

Adobe (2007a). Adobe director. `http://www.adobe.com/products/director/`.

Adobe (2007b). Adobe flash. `http://www.adobe.com/products/flash/`.

Bruder, S. and Wedeward, K. (2003). Robotics in the classroom. *IEEE Robotics & Automation Magazine*, 10(3):25–29.

Camurri, A., Hashimoto, S., Ricchetti, M., and et al (2000). Eyesweb: Toward gesture and affect recognition in interactive dance and music systems. *Computer Music Journal*, 24(1):57–69.

Ceccarelli, M. (2003). Robotic teachers' assistants. *IEEE Robotics & Automation Magazine*, 10(3):37–45.

Cycling74 (2007). Max/msp. `http://www.cycling74.com`.

da Costa, B. (2007). Beatriz da costa. `http://www.beatrizdacosta.net/index.php`.

f18institute (2007). Cellobot. `http://www.f18institut.org/f18-institut/f18-robotics.html`.

Fagin, B. (2003). Ada/mindstorms 3.0. *IEEE Robotics & Automation Magazine*, 10(2):19–24.

Fry, B. and Reas, C. (2007). Processing.org. `http://www.processing.org`.

Klassner, F. and Anderson, S. D. (2003). Lego mindstorms: not just for k-12 anymore. *IEEE Robotics Automation Magazine*, 10(2):12–18.

Lego Group (2006a). Lego mindstorms nxt bluetooth developer kit. `http://mindstorms.lego.com/Overview/NXTreme.aspx`.

Lego Group (2006b). Lego mindstorms nxt hardware developer kit. `http://mindstorms.lego.com/Overview/NXTreme.aspx`.

Lego Group (2007). Lego.com mindstorms nxt home. `http://mindstorms.lego.com`.

Moura, L. and Pereira, H. G. (2004). *Man + Robots : Symbiotic Art*.

Pisaturo, C. (2007). Works of carl pisaturo. `http://www.carlpisaturo.com/index.html`.

Puckette, M. (1996). Pure data: another integrated computer music environment. In *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41.

Ullanta (2007). Performance robotics. `http://www.ullanta.com/ullanta/`.

Wright, M. and Freed, A. (1997). Opensound control: A new protocol for communicating with sound synthesizers. In *Proceedings of the 1997 International Computer Music Conference*.