Integrazione di Raspberri PI con Lego RCX RIS

1. Abilitare Lego USB IR Tower

   (a) Collegare Tower

   (b) Controllare se il driver è stato caricato

```
pi@raspberrypi ~ $ ls -l /dev/usb/
total 0
crw-rw-rwT 1 root lego 180, 160 Jan 1 1970 legousbtower0
```

   (c) Settare i permessi d'accesso per i dispositivi *legousbtower* tramite una regola *udev*

   - Creare il file */etc/udev/rules.d/90-legotower.rules*
   - Aggiungere la regola nel file creato

```
ATTRS{idVendor}=="0694",ATTRS{idProduct}=="0001",
    MODE="0666",GROUP="lego"
```

   - Aggiungere l'utente *pi* nel gruppo *lego*

```
usermod -a -G lego pi
```

   (d) Dopo il riavvio l'utente *pi* potrà accedere ai dispositivi del tipo *Lego USB Tower* collegati

2. Installazione e impostazione del NQC per l'acceso a *legousbtower*

   (a) Scaricare e scompattare l'ultima versione del NQC [1]

```
mkdir nqc-3.1.r6 && cd nqc-3.1.r6
wget http://bricxcc.sourceforge.net/nqc/release/nqc-3.1.r6.tgz
tar xfz nqc-3.1.r6.tgz
cd ..
```

   (b) Scaricare e applicare il patch per abilitare l'acceso a USB

```
wget http://sourceforge.net/p/bricxcc/patches/_discuss/thread/
    00b427dc/b84b/attachment/nqc-01-Linux_usb_and_tcp.diff
patch -p0 < nqc-01-Linux_usb_and_tcp.diff
```

   Queste operazioni devono essere svolte nella directory in cui si trova la directory *nqc-3.1.r6*

   (c) Compilazione di NQC

```
cd nqc-3.1.r6
make
```

   Questa operazione richiede alcuni minuti

   (d) Alla fine della compilazione controlliamo che i binari siano stati creati

```
pi@raspberrypi ~/nqc-3.1.r6 $ ll bin/
total 704
-rwxr-xr-x 1 pi pi 12859 Nov 1 06:28 mkdata
-rwxr-xr-x 1 pi pi 703779 Nov 1 06:33 nqc
```

[1] al momento 3.1r6

(e) Installazione di NQC nel sistema.

```
sudo make install
```

3. Impostazioni iniziali con NQC

(a) Installazione del firmware

RCX lasciata senza alimentazione per qualche minuto perde il firmware, che in questo caso dovrebbe essere caricato nuovamente.

Caricare il firmware con NQC

```
pi@raspberrypi ~/rcx $ nqc -Susb:/dev/usb/legousbtower0 -firmware
    firm0328.lgo
Downloading firmware:..............
Current Version: 00030001/00030208
```

(b) Applicazione per il controllo dei motori.

```
pi@raspberrypi ~/rcx $ cat hellomsg.qnc
task main()
{
      while(true)
      {
            ClearMessage();
            until(Message() != 0);
            if(Message() == 1) { OnFwd(OUT_A + OUT_C);}
            if(Message() == 2) { OnRev(OUT_A + OUT_C);}
            if(Message() == 3) { Off(OUT_A + OUT_C);}
      }
}
```

Carichiamo e mettiamo in esecuzione il programma su RCX

```
nqc -Susb:/dev/usb/legousbtower0 -d hellomsg.qnc -pgm 3 -run
Downloading Program:...complete
Battery Level = 8.0 V
```

Da adesso è possibile controllare RCX inviandogli i messaggi. Sotto un esempio di messaggio di comando per andare avanti

```
nqc -Susb:/dev/usb/legousbtower0 -msg 1
```

E' possibile programmare fino a 255 messaggi. Il tempo di invio di un singolo messaggio è circa 0.5 secondi.

4. Installazione e impostazione di LeJOS su Raspberry Pi

(a) Scaricare l'ambiente LeJOS

```
mkdir -p /home/pi/rcx/lejos/
cd /home/pi/rcx/lejos
wget http://www.lejos.org/tools/lejos3/lejos.3.0.0-RC2.tar.gz
```

(b) Decomprimere l'archivio

```
tar xzf lejos.3.0.0-RC2.tar.gz
```

(c) Settare le variabili d'ambiente

```
export LEJOS_HOME=/home/pi/rcx/lejos
export PATH=$PATH:$LEJOS_HOME/bin
export CLASSPATH=$CLASSPATH:.:$LEJOS_HOME/lib/classes.jar
    :$LEJOS_HOME/lib/pcrcxcomm.jar
```

```
export RCXTTY=/dev/usb/legousbtower0
export JAVA_HOME=/usr/lib/jvm/jdk-7-oracle-armhf
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$LEJOS_HOME/bin
```

(d) Controllare i permessi degli eseguibili e se necessario resettarli

```
chmod +x /home/pi/rcx/lejos/bin/*
chmod +x /home/pi/rcx/lejos/release/*
```

(e) Compilare LeJOS.

i. Installare *ant*

```
sudo apt-get install ant
```

ii. Procedere con la compilazione

```
cd build/
ant
```

Questa operazione può richiedere alcuni minuti.

5. Abilitare RCX per l'utilizzo con LeJOS

(a) Collegare USB IR Tower con Raspberry Pi e accendere RCX.

(b) Caricare LeJOS firmware su RCX

```
cd /home/pi/rcx/lejos/bin/
pi@raspberrypi ~/rcx/lejos/bin $ ./firmdl
read firmware srec
  100%
Installing firmware
Loading native libs
Deleting firmware
Firmware deleted
Downloading firmware
  100%
Firmware downloaded
Unlocking firmware
Firmware unlocked
```

6. Semplice applicazione per controllare gli output su RCX con Raspberri Pi

Constants.java definisce i comandi di movimento

```
package resurrection.Constants;
public final class Constants {
public static final int Forward=1;
public static final int Backward=2;
public static final int Left=3;
public static final int Right=4;
public static final int Stop=5;
}
```

MainRCX.java è il main dell'applicazione che deve essere caricata su RCX

```
package resurrection.rcx;
import resurrection.Constants.Constants;
import josx.platform.rcx.*;

public class MainRCX {
   public static void main(String args[]) throws Exception {
      TowerRCX Canale = new TowerRCX();
```

```
        TextLCD.print("hello");
        while (true) {
            int command = Canale.receiveFromRPI();
            Canale.port.reset();
            LCD.showNumber(command);
            if (command == Constants.Forward) {
                Motor.A.setPower(7);
                Motor.C.setPower(7);
                Motor.A.forward();
                Motor.C.forward();
                TextLCD.print("go");
            }
            if (command == Constants.Backward) {
                Motor.A.setPower(2);
                Motor.C.setPower(2);
                Motor.A.backward();
                Motor.C.backward();
                TextLCD.print("bck");
            }
            if (command == Constants.Left) {
                Motor.A.backward();
                Motor.C.setPower(7);
                Motor.C.forward();
                TextLCD.print("lft");
            }
            if (command == Constants.Right) {
                Motor.C.backward();
                Motor.A.setPower(7);
                Motor.A.forward();
                TextLCD.print("rgt");
            }
            if (command == Constants.Stop) {
                TextLCD.print("stp");
                Motor.A.stop();
                Motor.C.stop();
            }
        }
    }
}
```

TowerRCX.java permette la comunicazione con Raspberry Pi tramite Tower

```
package resurrection.rcx;
import java.io.DataInputStream;
import josx.platform.rcx.*;
import josx.rcxcomm.RCXPort;

public class TowerRCX {
    public RCXPort port;
    private DataInputStream dis;

    public TowerRCX() {
        try {
            port = new RCXPort();
            dis = new DataInputStream(port.getInputStream());
        } catch (Exception e) {
            TextLCD.print("er1");
        }
    }

    public int receiveFromRPI() {
        int ricevutoDaRPI = 0;
        try {
```

```
22            ricevutoDaRPI = dis.readInt();
23         } catch (Exception e) {
24            TextLCD.print("er3");
25         }
26         return (ricevutoDaRPI);
27      }
28   }
```

MainRPI.java è l'applicazione eseguibile su RPI

```
1  package resurrection.rpi;
2
3  import resurrection.Constants.Constants;
4
5  public class MainRPI {
6     public static void main(String args[]) throws Exception {
7        TowerRPI canaleIr=new TowerRPI();
8        System.out.println("canale creato");
9        System.out.println("Forward");
10       canaleIr.sendToRCX(Constants.Forward);
11       System.out.println("Backward");
12       canaleIr.sendToRCX(Constants.Backward);
13       System.out.println("Left");
14       canaleIr.sendToRCX(Constants.Left);
15       System.out.println("Right");
16       canaleIr.sendToRCX(Constants.Right);
17       System.out.println("Stop");
18       canaleIr.sendToRCX(Constants.Stop);
19    }
20 }
```

TowerRPI permette la comunicazione con RXT tramite Tower

```
1  package resurrection.rpi;
2  import java.io.DataOutputStream;
3  import java.io.OutputStream;
4  import josx.rcxcomm.RCXPort;
5
6  public class TowerRPI {
7     DataOutputStream dos;
8     private RCXPort port;
9     public TowerRPI(){
10       try {
11          port = new RCXPort("/dev/usb/legousbtower0");
12          OutputStream os = port.getOutputStream();
13          dos = new DataOutputStream(os);
14       }
15       catch (Exception e) {
16          System.out.println(e);
17       }
18    }
19    public void sendToRCX(int comando){
20       try {
21          dos.writeInt(comando);
22          dos.flush();
23       }
24       catch (Exception e) {
25          System.out.println(e);
26       }
27    }
28 }
```

(a) Compilazione e caricamento dell'applicazione su RXC

```
TowerRP@raspberrypi ~/rcx $ lejos resurrection/rcx/MainRCX.
MainRCX.class MainRCX.java
pi@raspberrypi ~/rcx $ lejos resurrection/rcx/MainRCX
linking...
downloading...
read binary
  100%
download binary
Loading native libs
download program
  100%
  100%
```

Per mettere in esecuzione l'applicazione su RXC schiacciare il pulsante *Run*.

(b) Compilazione e esecuzione dell'applicazione su Raspberry pi

```
pi@raspberrypi ~/rcx $ javac resurrection/rpi/MainRPI.java
pi@raspberrypi ~/rcx $ java resurrection.rpi.MainRPI
Loading native libs
canale creato
Forward
Backward
Left
Right
Stop
```

Anche sul display di RCX devono essere visibili i comandi durante l'esecuzione.

Link utili

- http://pbrick.info/2013/10/configuring-the-lego-usb-tower-on-linux/
- http://xed.ch/help/lego.html
- http://www.lejos.org/