TEST DEI SENSORI A ULTRASUONI

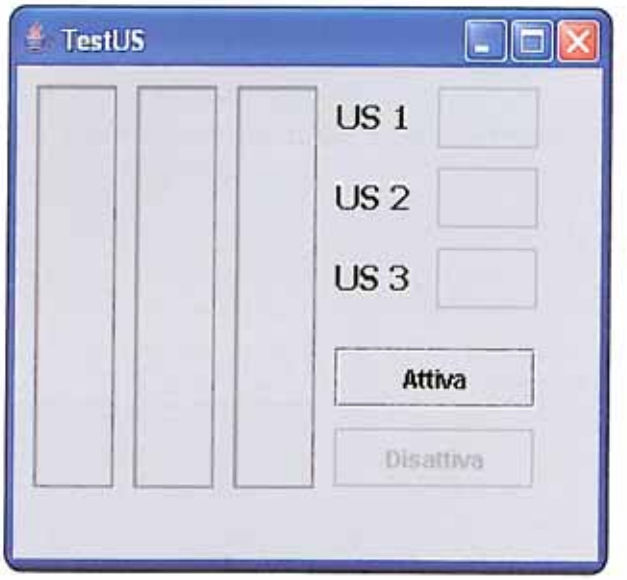
Il programma in Java presentato in questo fascicolo permette di monitorare, mediante un'apposita interfaccia grafica, i dati forniti dai sensori a ultrasuoni di I-Droid01. L'interfaccia presenta una modalità di visualizzazione 'qualitativa', tramite barre graduate colorate, e una 'quantitativa', tramite valori numerici.

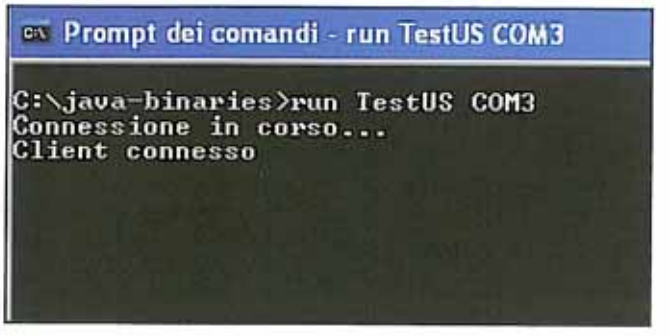
In questo nuovo programma di esempio proposto nelle prossime pagine, sfrutteremo le librerie Java per accedere ai sensori a ultrasuoni di I-Droid01. In particolare sarà sviluppata una semplice interfaccia grafica che permetterà di 'monitorare' quanto rilevato dai sensori che, ricordiamo, sono localizzati nella parte anteriore del marsupio del robot. L'interfaccia è costituita da tre barre colorate che permettono di visualizzare in maniera 'qualitativa' i dati forniti dai tre sensori a ultrasuoni. Tali dati corrispondono alla distanza in centimetri tra il robot ed eventuali ostacoli. Per avere una misura 'quantitativa' della distanza, invece, i dati rilevati sono presentati numericamente tramite tre apposite celle poste sul lato destro dell'interfaccia. A completamento di quest'ultima sono presenti due pulsanti, rispettivamente per attivare e disattivare le letture dei dati dei sensori a ultrasuoni. Il codice del programma è strutturato in due classi: **USControl** fornirà le primitive per leggere i registri associati ai sensori, mentre **TestUS** realizzerà l'interfaccia grafica del programma di monitoraggio. Per quanto riguarda i dati rilevati dai sensori a ultrasuoni, ricordiamo come le misurazioni siano fortemente dipendenti dalla forma degli ostacoli. Con ostacoli perpendicolari all'asse dei sensori e ben delineati (come le pareti di una stanza), I-Droid01 è in grado di rilevare con una buona precisione anche quelli posti a oltre un metro di distanza. La distanza minima rilevabile, invece, è di poco superiore a 20 centimetri.

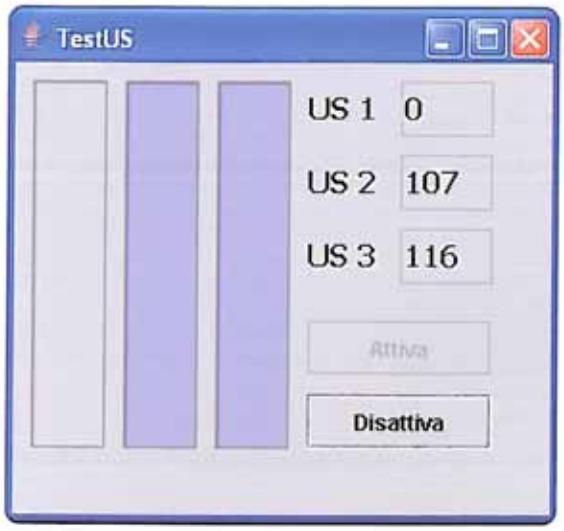
USCONTROL

La prima delle due classi in cui è strutturato il programma è **USControl**. Essa fornisce e implementa i metodi per l'accesso ai registri di I-Droid01 relativi alla gestione dei sensori a ultrasuoni. Per prima cosa sono presenti le solite dichiarazioni di pacchetto e di importazione: notiamo subito come compaiano i pacchetti relativi al modulo Base. Ricordiamo, infatti, che i sensori a ultrasuoni sono controllati proprio da questa specifica scheda elettronica. La classe fornisce una coppia di metodi per attivare e disattivare i sensori: si tratta, rispettivamente,

Sotto, il pannello grafico per il monitoraggio dei sensori a ultrasuoni prima dell'avvio del rilevamento dei dati, come si può notare dall'assenza di valori. Più in basso la classica schermata del prompt subito dopo l'avvio del programma.





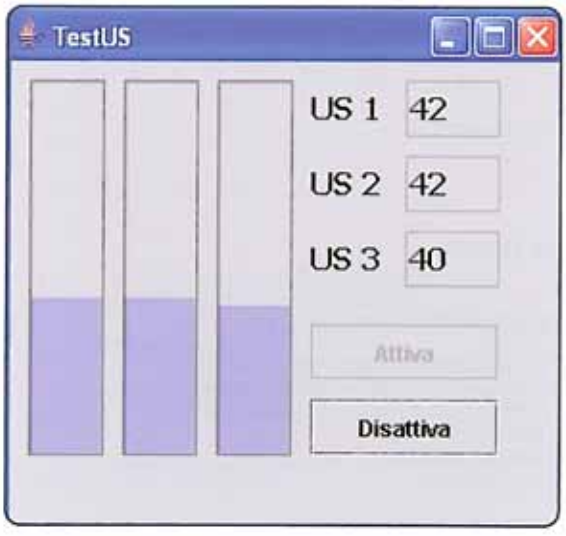


Sopra, la schermata del programma mentre è attivo il monitoraggio dei sensori. Il primo sensore non rileva nessun ostacolo, a differenza degli altri due sensori.

di **activateUS** e **deactivateUS**. I due metodi scrivono un apposito valore numerico nel registro **BASIC_BEHAVIOR** tramite la consueta istruzione **writeRegister**. Seguono alcuni metodi per leggere i dati rilevati dai tre ricevitori a ultrasuoni collocati nel marsupio di I-Droid01. Più precisamente, i metodi che si occupano di ciò sono quattro: i primi tre (**getUS1**, **getUS2** e **getUS3**) restituiscono il valore misurato da ogni singolo ricevitore, mentre il quarto (**getAll**), invece, restituisce l'intera terna di valori sotto forma di un unico array numerico. In caso di errore durante la lettura dei registri i metodi descritti restituiscono il valore -1. L'accesso ai dati misurati dai sensori a ultrasuoni avviene mediante una semplice lettura dei registri associati (**ULTRASOUND1**, **ULTRASOUND2**, **ULTRASOUND3**). Il metodo **getAll** effettua comunque una lettura del registro **ULTRASOUND1**, ma anziché leggere un singolo valore ne legge tre, visto che i registri associati ai singoli sensori sono adiacenti tra loro.

TESTUS

L'interfaccia grafica del programma è realizzata dalla classe **TestUS**. Dopo le consuete dichiarazioni sono presenti gli attributi e i metodi della classe. Tra questi vi è il **main**, che si occupa di istanziare un oggetto di tipo **TestUS** e avviare l'esecuzione dell'applicazione. Al solito, il metodo controlla che venga fornita, come parametro di avvio del programma, la porta seriale virtuale per comunicare con il robot. Il costruttore della classe, invece, si occupa della creazione dell'interfaccia grafica. Per istanziare i diversi elementi, il costruttore utilizza un metodo ausiliario, **createUSPanel**. Alle barre verticali, istanze della classe **JProgressBar**, è associato un intervallo numerico compreso tra 0 e 100, nonostante il range dei sensori a ultrasuoni sia più elevato: i valori 'esatti' sono, in ogni caso, riportati sui tre campi testuali nella parte destra dell'interfaccia grafica. I due pulsanti 'Attiva' e 'Disattiva' servono per avviare e interrompere la lettura dai registri interni associati ai sensori e il conseguente aggiornamento degli elementi grafici. A tale scopo è presente un apposito thread, **RefreshValue**, che si occupa di svolgere tutte le operazioni necessarie. **RefreshValue** è realizzato mediante una classe interna che implementa l'interfaccia **Runnable**. Il codice di questa classe è costituito da un ciclo **while** eseguito solo nel caso in cui la variabile booleana **active** sia settata sul valore 'true': il settaggio di questa variabile viene eseguito quando l'utente preme il pulsante 'Attiva'. Quando viene premuto l'altro pulsante, 'Disattiva', la variabile viene invece impostata sul valore 'false'. Quando il ciclo **while** è in esecuzione, la lettura dei registri di I-Droid01 relativi ai sensori a ultrasuoni viene eseguita con una cadenza pari a 200 millisecondi, come è possibile notare dall'istruzione **Thread.sleep(200)**, che sospende temporaneamente il thread. Gli ultimi metodi della classe **TestUS** sono i consueti **connetti** e **disconnetti**. Per quanto riguarda il primo non ci sono differenze rispetto al codice dei programmi presentati nei fascicoli precedenti. Il secondo metodo, invece, si occupa di settare il valore della variabile **active** a 'false', prima di attuare la disconnessione tra il computer e il robot.



Sopra, un'altra schermata del programma TestUS. Questa volta tutti e tre i sensori rilevano la presenza di un ostacolo, rispettivamente alla distanza di 42, 42 e 40 centimetri.

ESEMPI DI PROGRAMMAZIONE

Esempio di codice Java per l'implementazione di una classe per l'accesso ai sensori a ultrasuoni e di una classe per la realizzazione di un'interfaccia grafica per il monitoraggio dei dati rilevati da tali sensori. Il codice della prima classe (USControl), è molto semplice e contiene alcuni metodi per la lettura dei registri associati ai sensori a ultrasuoni, come il metodo getUS. Sono inoltre presenti due metodi per attivare e disattivare i sensori. Per quanto riguarda la seconda classe (TestUS), invece, oltre ai soliti metodi e attributi per la gestione degli elementi grafici, è possibile notare l'esistenza di due classi interne, ButtonHandler e RefreshValue. La prima è adibita alla gestione degli eventi associati alla pressione dei due pulsanti, la seconda, invece, si occupa della lettura dei valori rilevati dai sensori a ultrasuoni, aggiornando di conseguenza gli elementi grafici dell'interfaccia.

CLASSE USCONTROL

```
package communication.examples;

import communication.handler.internal.InternalHandler;
import communication.handler.internal.InternalModule;
import communication.handler.internal.BaseData;
import communication.handler.internal.BaseRegister;
import java.io.IOException;

public class USControl {

  private InternalHandler internal;

  public USControl(InternalHandler internal) {
      this.internal = internal;
  }

  public boolean activateUS() { [...] }

  public boolean deactivateUS() { [...] }

  public int getUS1() { [...] }

  public int getUS2() { [...] }

  public int getUS3() { [...] }

  public int[] getAll() { [...] }

} // class USControl
```

CLASSE TESTUS

```
package communication.examples;

import communication.handler.ProtocolHandler;
import communication.handler.internal.InternalHandler;
import communication.transport.ConnectionProvider;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TestUS extends JFrame {

  private static String portName;
  private ProtocolHandler protocol;
  private InternalHandler internal;
  private USControl us;
  JPanel usPanel;
  JProgressBar bar1;
  JProgressBar bar2;
  JProgressBar bar3;
  JLabel label1;
  JLabel label2;
  JLabel label3;
  JTextField us_value1;
  JTextField us_value2;
  JTextField us_value3;
  JButton activate;
  JButton deactivate;
  boolean active;
  Thread t;

  public static void main(String[] args) { [...] }

  public TestUS () { [...] }

  JPanel createUSPanel() { [...] }

  private class ButtonHandler implements ActionListener {
      public void actionPerformed(ActionEvent e) {
          if (e.getSource() == activate) {
              us.activateUS();
              active = true;
              activate.setEnabled(false);
              deactivate.setEnabled(true);
          } else if (e.getSource() == deactivate) {
              active = false;
              us.deactivateUS();
              activate.setEnabled(true);
              deactivate.setEnabled(false);
              bar1.setValue(0);
              bar2.setValue(0);
              bar3.setValue(0);
              us_value1.setText("");
              us_value2.setText("");
              us_value3.setText("");
          }
      }

  } // class ButtonHandler

  private class RefreshValue implements Runnable {
      int[] v = new int[3];
      public void run() {
          while(true) {
              if (active) {
                      v = us.getAll();
                      bar1.setValue(v[0]);
                      bar2.setValue(v[1]);
                      bar3.setValue(v[2]);
                      us_value1.setText(Integer.toString(v[0]));
                      us_value2.setText(Integer.toString(v[1]));
                      us_value3.setText(Integer.toString(v[2]));
                      try {
                              Thread.sleep(200);
                      } catch (Exception e) {
                              e.printStackTrace();
                      }
              }
          }
      }

  } // class RefreshValue;

  private boolean connetti() { [...] }

  private void disconnetti() { [...] }

} // TestUS
```

ESEMPI DI PROGRAMMAZIONE

Codice di implementazione dei metodi della classe USControl. Essa fornisce i metodi per accedere in lettura ai registri di I-Droid01 relativi alla gestione dei sensori a ultrasuoni. I dati forniti corrispondono alla distanza in centimetri dal primo ostacolo 'visibile' da ognuno dei tre ricevitori a ultrasuoni. I metodi che si occupano della lettura sono quattro (getUS1, getUS2, getUS3 e getAll). Prima di utilizzare questi metodi è però necessario attivare i sensori: per fare questo è sufficiente invocare il metodo activate. Qualora si verificassero degli errori durante l'accesso ai registri, i metodi restituiscono il valore -1. Per disattivare i sensori a ultrasuoni, qualora non fosse più necessario effettuare le misurazioni, è necessario invocare il metodo opposto, deactivate.

METODI DELLA CLASSE USCONTROL

```
public boolean activateUS() {

    int[] buf = new int[1];
    buf[0] = BaseData.ENABLE_ULTRASOUND;

    try {

        internal.writeRegister(InternalModule.BASE,
BaseRegister.BASIC_BEHAVIOR, buf);

    } catch (Exception e) {

        e.printStackTrace();
        return false;

    }

    return true;

} // activateUS


public boolean deactivateUS() {

    int[] buf = new int[1];
    buf[0] = 0;

    try {

        internal.writeRegister(InternalModule.BASE,
        BaseRegister.BASIC_BEHAVIOR, buf);

    } catch (Exception e) {

        e.printStackTrace();
        return false;

    }

    return true;

} // deactivateUS


public int getUS1() {

    int[] buf = new int[1];

    try {

        internal.readRegister(InternalModule.BASE,
BaseRegister.ULTRASOUND1, buf);

    } catch (Exception e) {

        e.printStackTrace();
        return -1;

    }

    return buf[0];

} // getUS1


public int getUS2() {

    int[] buf = new int[1];

    try {

        internal.readRegister(InternalModule.BASE,
BaseRegister.ULTRASOUND2, buf);

    } catch (Exception e) {

        e.printStackTrace();
        return -1;
    }

    return buf[0];

} // getUS2


public int getUS3() {

    int[] buf = new int[1];

    try {

        internal.readRegister(InternalModule.BASE,
BaseRegister.ULTRASOUND3, buf);

    } catch (Exception e) {

        e.printStackTrace();
        return -1;

    }

    return buf[0];

} // getUS3


public int[] getAll() {

    int[] buf = new int[3];

    try {

        internal.readRegister(InternalModule.BASE,
BaseRegister.ULTRASOUND1, buf);

    } catch (Exception e) {

        e.printStackTrace();
        buf[0]=-1;
        buf[1]=-1;
        buf[2]=-1;
        return buf;

    }

    return buf;

} // getAll
```

ESEMPI DI PROGRAMMAZIONE

Codice di implementazione dei metodi della classe USControl. Il codice si occupa della realizzazione dell'interfaccia grafica del programma. Essa realizza il pannello grafico con le tre barre colorate progressive e le caselle numeriche per la visualizzazione delle distanze misurate dai tre sensori a ultrasuoni. Sono inoltre presenti i soliti metodi connetti e disconnetti per attivare e disattivare, rispettivamente, la connessione tra il robot e il PC.

METODI DELLA CLASSE USCONTROL

```
public static void main(String[] args) {

    if (args.length < 1) {
        System.err.println("Specifica la porta da usare per la
connessione (es. COM5)");
        return;
    }

    portName = args[0];
    TestUS gui = new TestUS();

} // main

public TestUS () {

    super("TestUS");
    addWindowListener(
        new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                disconnetti();
                System.exit(0);
            }
        });
    setBounds(0,0,300,280);
    usPanel =  createUSPanel();
    getContentPane().add(usPanel);
    if (!connetti()) System.exit(0);
    us = new USControl(internal);
    t = new Thread(new RefreshValue());
    active = false;
    t.start();
    setVisible(true);

} // TestUS


JPanel createUSPanel() {

    JPanel panel = new JPanel();
    panel.setLayout(null);
    bar1 = new JProgressBar(JProgressBar.VERTICAL, 0, 100);
    bar1.setStringPainted(false);
    bar1.setBounds(10,10,40,200);
    bar2 = new JProgressBar(JProgressBar.VERTICAL, 0, 100);
    bar2.setStringPainted(false);
    bar2.setBounds(60,10,40,200);
    bar3 = new JProgressBar(JProgressBar.VERTICAL, 0, 100);
    bar3.setStringPainted(false);
    bar3.setBounds(110,10,40,200);
    label1 = new JLabel("US 1");
    label1.setFont(new Font("Tahoma", Font.BOLD, 16));
    label1.setBounds(160,10,40,30);
    label2 = new JLabel("US 2");
    label2.setFont(new Font("Tahoma", Font.BOLD, 16));
    label2.setBounds(160,50,40,30);
    label3 = new JLabel("US 3");
    label3.setFont(new Font("Tahoma", Font.BOLD, 16));
    label3.setBounds(160,90,40,30);
    us_value1 = new JTextField();
    us_value1.setFont(new Font("Tahoma", Font.BOLD, 16));
    us_value1.setBounds(210,10,50,30);
    us_value1.setEditable(false);
    us_value2 = new JTextField();
    us_value2.setFont(new Font("Tahoma", Font.BOLD, 16));
    us_value2.setBounds(210,50,50,30);
    us_value2.setEditable(false);
    us_value3 = new JTextField();
    us_value3.setFont(new Font("Tahoma", Font.BOLD, 16));
    us_value3.setBounds(210,90,50,30);
    us_value3.setEditable(false);
    ButtonHandler buttonHandler = new ButtonHandler();
    activate = new JButton("Attiva");
    activate.setBounds(160,140,100,30);
    activate.addActionListener(buttonHandler);
    deactivate = new JButton("Disattiva");
    deactivate.setBounds(160,180,100,30);
    deactivate.addActionListener(buttonHandler);
    deactivate.setEnabled(false);
    panel.add(bar1);
    panel.add(bar2);
    panel.add(bar3);
    panel.add(label1);
    panel.add(label2);
    panel.add(label3);
    panel.add(us_value1);
    panel.add(us_value2);
    panel.add(us_value3);
    panel.add(activate);
    panel.add(deactivate);
    return panel;

} // createUSPanel


private boolean connetti() {

    ConnectionProvider provider =
new SerialJavaxCommProvider(portName,9600);
    protocol = new ProtocolHandler(provider, 9600);
    internal = new InternalHandler();
    protocol.addCapabilityHandler(internal);
    System.out.println("Connessione in corso...");
    try {
        if (!protocol.connect()) {
                System.err.println("Impossibile connettersi!");
                return false;
        }
        if (!protocol.setup(5000)) {
            System.err.println("Settaggio della connessione fallito!");
            protocol.disconnect(true);
            return false;
        }
    } catch (Exception e) {
            e.printStackTrace();
            return false;
    }
    System.out.println("Client connesso");
            return true;
} // connetti


private void disconnetti() {

    System.out.println("Disconnessione in corso...");
    try {
        active = false;
        protocol.disconnect(false);
    } catch (Exception e) {
        e.printStackTrace();
        return;
    }
    System.out.println("Client disconnesso");

} // disconnetti
```