Series 40 communicating with Arduino using Bluetooth

This article explains how to communicate with an Arduino board using a Series 40 device through Bluetooth.

Introduction

This previous post demonstrated how to implement a Windows Phone 8 that communicates with an Arduino board through Bluetooth; Now, we are going to do the same using an Asha 310.

In this post I'll show the J2ME code needed to access the Arduino and a complete example. The communication architecture, the arduino code and the schematic are exactly the same as shown in this post, no changes are needed and any Series 40 devices with the SPP Bluetooth profile available is able to access the Arduino board through Bluetooth.

As almost all the work was done previously in this post, some of the steps are just copies of the original section with small updates. If you already did the job with Windows Phone 8, you can jump over next sections and go to the Series 40 code update subsection.

If you are new to Arduino, it is an open-source board that accept a lot of sensors and hardware extensions. So, it is possible to communicate with an Arduino board in many ways such as Ethernet, USB, Bluetooth, etc.

As described in Arduino's web site, "Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software". A quick search for "Arduino" on the Internet yields that many cool projects are using it.

Preparing hardware

This section is a copy of this one.

Arduino Uno

In this article, I used Arduino Compatible UNO Starter Kit which came with a lot of wires and LEDs.

Bluetooth module

JY-MCU Arduino Bluetooth Wireless Serial Port Module is a Bluetooth module which is used in this article. In spite of its low price, it works fine. But I believe that you can use any Bluetooth module that works with SoftwareSerial.h library.

Proximity sensor

The proximity sensor detects a near body (30cm at maximum). As the Bluetooth communication doesn't depends on this proximity sensor, you can use any other kind of sensor in your project. I just used this one because I already have it. But I could use a temperature sensor, a luminosity sensor, etc. A small change is required in the project should you want to use any of them.

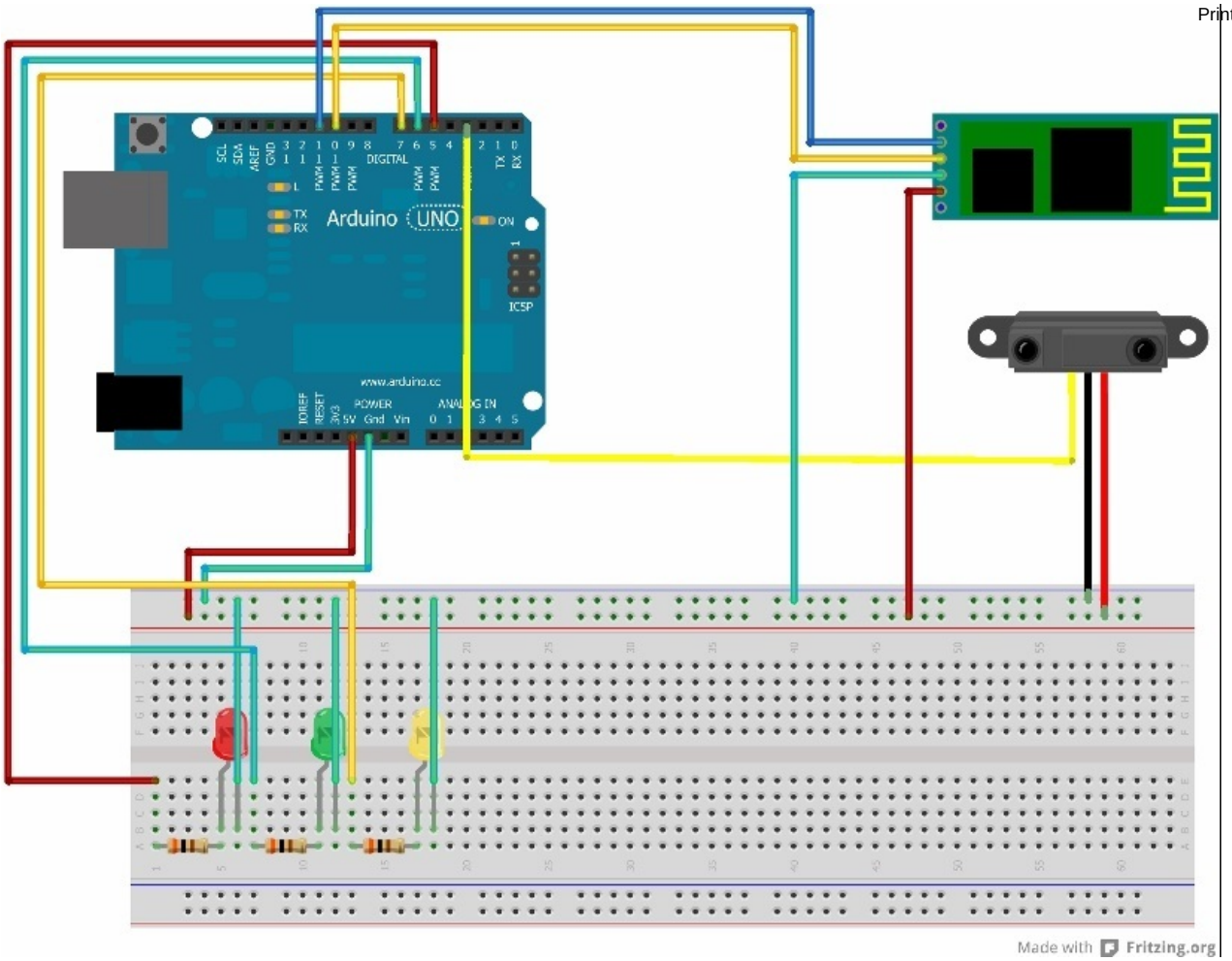
Arduino Infrared Obstacle Avoidance Detection Photoelectric Sensor is used as the proximity sensor in this article.

Configuring Arduino

In order to configure the Arduino development environment on your computer (Windows, Linux or Mac OS X), please visit Getting Started with Arduino.

Once you get all the needed hardware, they must be configured as shown in the following image. In case of any doubt about Arduino, please refer to the Arduino web site (http://www.arduino.cc).

(C) Copyright Nokia 2014. All rights reserved.



Parts list

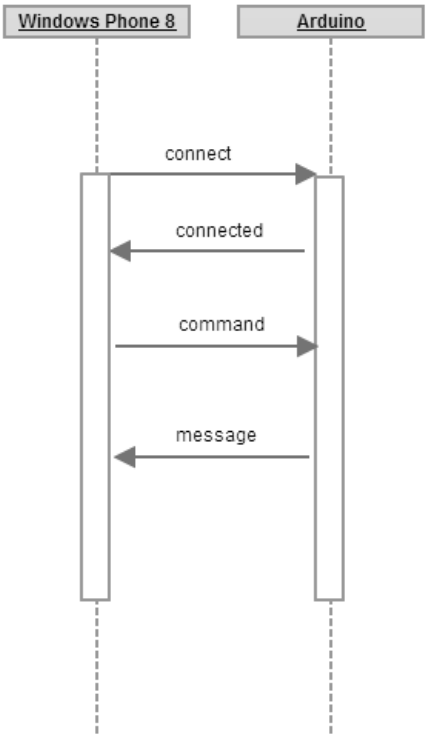
- Arduino Uno
- JY-MCU Arduino Bluetooth Wireless Serial Port Module
- Arduino Infrared Obstacle Avoidance Detection Photoelectric Sensor
- 15 wires
- 3 LEDs (Red, Green and Yellow)
- 3 Resistors (300 &Omega)
- 1 Protoboard

Architecture

This section is a copy of this one, only some details about the J2ME code were updated.

The communication flow is described below. Since the article focuses on communication part, and to make the things simple and clear, the commands and messages are of type string. If you want to send integer, float or any other type, the code and protocol must be improved.

(C) Copyright Nokia 2014. All rights reserved.



As shown in the diagram sequence, the Series 40 device (Windows Phone 8 in the diagram) and Arduino communicate each other through commands (Series 40 --> Arduino) and messages (Series 40 <-- Arduino). These commands/messages are of type string (with 255 bytes at maximum).

- Command - Consists of a header of one byte to inform the size of the current command (which limits the command size to 255 characters) and a body that is the command itself.
- Message - Consists of a header of one byte to inform the size of the current message (which limits the message size to 255 characters) and a body that is the message itself.

There are some core codes on both sides; Series 40 and Arduino; that you can reuse in other projects without changes.

The Arduino core code are the loop() and the sendMessage(char* message) functions, but you must give your own implementation of the sendPeriodicMessages() and processCommand(char* command) functions according to what you want to do. You must also include the "SofwareSerial.h" library in your project and create the related variables and constants to control the serial communication. The Arduino core code is exactly the same of the Windows Phone 8 version.

Series 40 code update

The J2ME core code are the `ConnectionManager` class and the `ConnectionListener` interface. The `ConnectionManager` class is used to connect, send commands and receive messages to/from the Arduino. The messages are received through the `ConnectionListener` interface that must be implemented by the developer.

Example code

In order to prove that this implementation works fine (The architecture has already proved in this post), one example was implemented. This J2ME example does the same as the Windows Phone 8 example.

- The user can turn on/turn off any specific led just on click;
- The user will be notified when someone/something is detected by the proximity sensor on the board;
- After detect someone, the J2ME application automatically turn on one specific led (the red one);

In the section Complete example, you can watch this example running or download the source code.

J2ME core code

ConnectionManager class

The `MessageReceiver` class is also declared here, but the developer doesn't have to care about it. It is only used by the `ConnectionManager` class.

(C) Copyright Nokia 2014. All rights reserved.

```
/**
 * Class to control the bluetooth connection to the Arduino.
 * @author Marcos
 */
public class ConnectionManager implements DiscoveryListener {

 /**
  * Local device.
  */
 private LocalDevice localDevice;

 /**
  * Discovery agent.
  */
 private DiscoveryAgent discoveryAgent;

 /**
  * Stream connection.
  */
 private StreamConnection connection;

 /**
  * Connection listener.
  */
 private ConnectionListener connectionListener;

 /**
  * Message receiver.
  */
 private MessageReceiver messageReceiver;

 /**
  * Device name.
  */
 private String deviceName;

 /**
  *
  * @param midlet
  * @throws BluetoothStateException
  */
 public ConnectionManager() throws BluetoothStateException {
  this.localDevice = LocalDevice.getLocalDevice();
  this.discoveryAgent = this.localDevice.getDiscoveryAgent();
 }

 /**
  * Set the connection listener.
  * @param listener
  */
 public void setConnectionListener(ConnectionListener listener) {
  this.connectionListener = listener;
 }

 /**
  * Connect to the device with the given name.
```


(C) Copyright Nokia 2014. All rights reserved.

```
   * @param deviceName Device name.
   */
  public void connect(String deviceName) {
   try {
    this.deviceName = deviceName;
    //Try to found the device in the cache list.
    RemoteDevice[] cachedDevices = discoveryAgent.retrieveDevices(DiscoveryAgent.CACHED);
    if(cachedDevices != null) {
     for (int i = 0; i < cachedDevices.length; i++) {
      RemoteDevice remoteDevice = cachedDevices[i];
      if(remoteDevice.getFriendlyName( false ).equals(deviceName)) {
       //Device found in the cached list.
       deviceDiscovered(remoteDevice, null);
       return;
      }
     }
    }
    //No device found in cache list. Start discovery.
    discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
   } catch (BluetoothStateException e) {
    connectionListener.notifyError(e);
   } catch (IOException e) {
    connectionListener.notifyError(e);
   }
  }

  /**
   * Cancel the current connection.
   */
  public void cancelConnection() {
   discoveryAgent.cancelInquiry(this);
  }

  public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {
   try {
    if(btDevice.getFriendlyName( false ).equals(deviceName)) {
     connectionListener.deviceFound(btDevice);
     connection = (StreamConnection)
Connector.open("btspp://"+btDevice.getBluetoothAddress()+":1");
     messageReceiver = new MessageReceiver(connection, connectionListener);
     messageReceiver.start();
    }
   } catch (IOException e) {
    connectionListener.notifyError(e);
   }
  }

  /**
   * Send command to the Arduino through bluetooth.
   * @param command The sent command.
   */
  public void sendCommand(String command) {
   try {
    OutputStream os = connection.openOutputStream();
    os.write(command.length());
    os.write(command.getBytes());
    os.close();
```


(C) Copyright Nokia 2014. All rights reserved.

```
  } catch (IOException e) {
   connectionListener.notifyError(e);
  }
 }

 public void inquiryCompleted(int discType) {
  // No implementation required
 }

 public void serviceSearchCompleted(int transID, int respCode) {
  // No implementation required
 }

 public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
  // No implementation required
 }
}

class MessageReceiver extends Thread {

 private InputStream inputStream;

 private ConnectionListener connectionListener;

 private boolean mustRun = false;

 public MessageReceiver(StreamConnection connection, ConnectionListener listener) throws
IOException {
  inputStream = connection.openInputStream();
  connectionListener = listener;
 }

 public void run() {
  try {
   this.mustRun = true;
   while (mustRun) {
    // Waiting for message size
    int messageSize = inputStream.read();
    byte[] messageArray = new byte[messageSize];
    for(int i = 0;i < messageArray.length;i++) {
     messageArray[i] = (byte)inputStream.read();
    }
    String message = new String(messageArray);
    connectionListener.messageReceived(message);
   }
  } catch (Exception e) {
   connectionListener.notifyError(e);
  }
 }

 public void stop() {
  this.mustRun = false;
  this.interrupt();
 }
}
```


(C) Copyright Nokia 2014. All rights reserved.

ConnectionListener interface

The `ConnectionListener` interface must be implemented by the developer in order to receive incoming messages, error notifications and about device connection accomplishment.

```
public interface ConnectionListener {

 /**
  * Fired when a new device is found.
  * @param btDevice
  */
 public void deviceFound(RemoteDevice btDevice);

 /**
  * The message is received.
  * @param message The message
  */
 public void messageReceived(String message);

 /**
  * Notify an error.
  * @param t
  */
 public void notifyError(Throwable t);
}
```

Arduino core code

The Arduino core code is exactly the same of the Windows Phone 8 post. No changes are needed.

```
#include <SoftwareSerial.h>

const int TX_BT = 10;
const int RX_BT = 11;

SoftwareSerial btSerial(TX_BT, RX_BT);

//Frequency to send periodic messages to Windows Phone, in milliseconds.
//Core code.
const unsigned long periodicMessageFrequency = 5000;
unsigned long time = 0;

//Process the incoming command from Windows Phone.
//It should be changed according to what you want to do.
void processCommand(char* command) {
}

//Send a message back to the Windows Phone.
//Is can't be changed.
void sendMessage(char* message) {
  int messageLen = strlen(message);
  if(messageLen < 256) {
    btSerial.write(messageLen);
    btSerial.print(message);
  }
```


(C) Copyright Nokia 2014. All rights reserved.

```
}

//Send a set of periodic messages to the Windows Phone.
//It should be changed according to what you want to do.
//This message could be a sensor data, like a thermometer data.
void sendPeriodicMessages() {
}

//Setup Arduino function
void setup() {
  Serial.begin(9600);
  Serial.println("USB Connected");
  btSerial.begin(9600);
}

//Loop Arduino function
//It can't be changed
void loop() {
  if(btSerial.available()) {
      int commandSize = (int)btSerial.read();
      char command[commandSize];
      int commandPos = 0;
      while(commandPos < commandSize) {
        if(btSerial.available()) {
          command[commandPos] = (char)btSerial.read();
          commandPos++;
        }
      }
      command[commandPos] = 0;
      processCommand(command);
  }
  unsigned long currentTime = millis();
  if((currentTime - time) > periodicMessageFrequency) {
    sendPeriodicMessages();
    time = currentTime;
  }
}
```

Complete example

If you already have the Arduino board configured as described in the previous section, you can download the complete example and test it.

Source list

- The J2ME code: (here)
- Arduino code: (here)

If you just want to see the example running, watch the below video.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.