TECHNICAL UNIVERSITY OF CLUJ-NAPOCA ROMANIA

Arduino – Android sensor data communication

Stancu Gabriel – Iulian
Technical University of Cluj – Napoca
Design with Microprocessors– 2020/21
Teaching assistant: Balazs Barna Moldovan

Arduino – Android sensor data communication

Contents

1. Introduction

The aim of this project was to simulate a wireless – controlled thermostat, using Arduino components, an Android device and the communication between them achieved through Bluetooth. The target was to create a user application that handles eventual exceptions during runtime, but also ensures communication with the Arduino board (the part of the project that simulates the thermostat) in a bug – free manner. For this purpose, both an Android application needed to be implemented, but also an Arduino application was developed for the Arduino controller's internal logic programming.

The Arduino board collects data about the temperature of the room it is placed in and sends it through Bluetooth to the Android device, which displays the temperature in a user – friendly interface. The user has the possibility of adjusting the temperature from his phone, command which is sent back through Bluetooth to the Arduino board and a simulation of the thermostat setting is performed by the board.

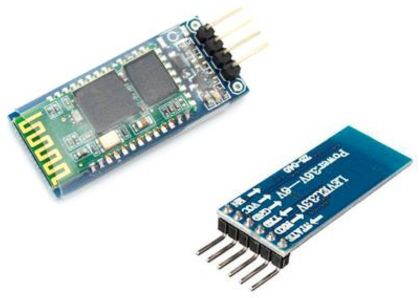
2. Design

Based on the previous specifications, the following modules and components were used:

- An Android device (for this project, a LG G4, Android version 6.0)

- An Arduino UNO board

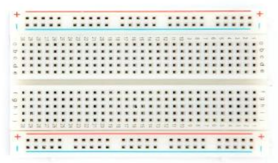
- A HC-05 Bluetooth module



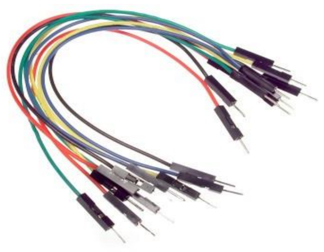
- A KY-028 Digital Temperature Sensor

- A SG90 Servomotor

- A breadboard



- Wires



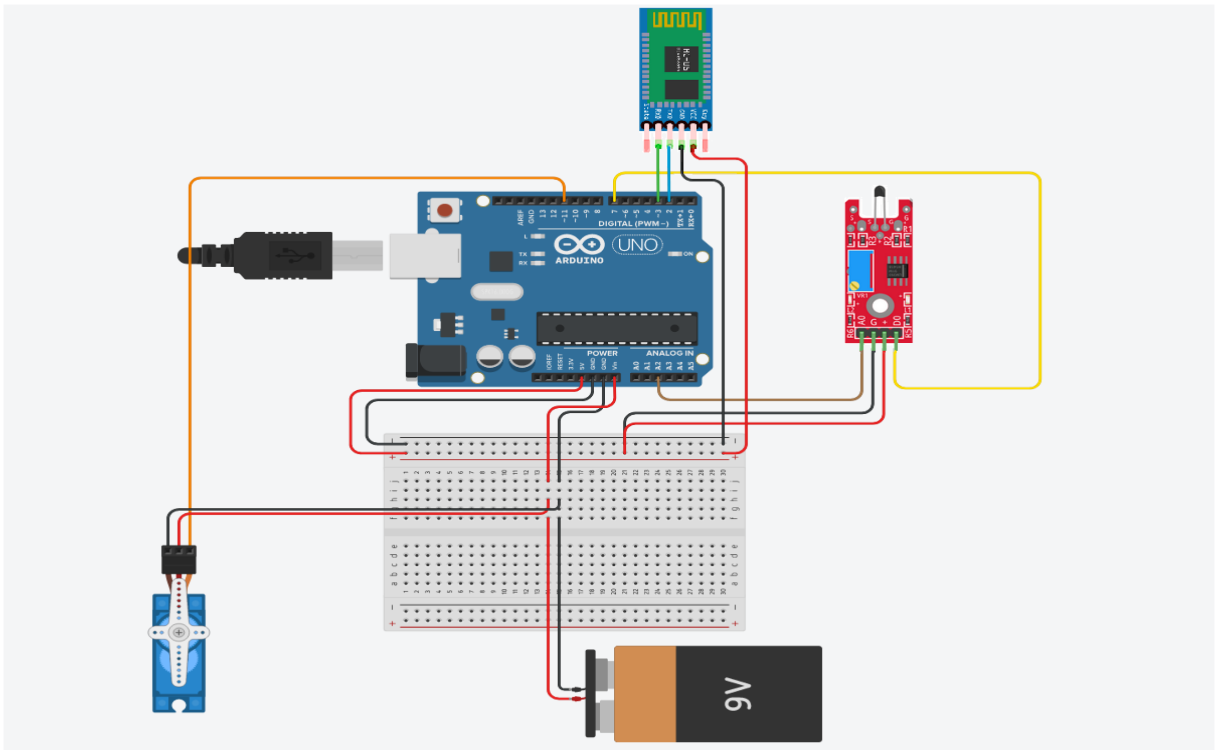
- A 9V battery

For achieving our target, the previously mentioned components will be used as follows: the temperature sensor will collect data and send it to the Arduino board, which uses the Bluetooth module to export the data to eventual receivers (the Android device in our case). The Android device displays the data to the user in a friendly interface and allows him to adjust the temperature in the room from the same interface. The Bluetooth module also receives the adjusting command, sends it to the Arduino board, which commands the servomotor to rotate at a desired angle, to simulate the setting of an actual thermostat. The breadboard is used for making it easier to connect the components, and the 9V battery is used to ensure enough power for the whole circuit, as the motor and the Bluetooth module together need a considerable amount of power.

For the Android part, we will use the C# programming language, inside the Xamarin Android framework developed by Microsoft.

3. Implementation

For the Arduino part, the circuit is the following:



The configuration is quite standard, with the Bluetooth module and the temperature sensor being connected according to their pinout. As previously mentioned, the 9V battery is used for supplying extra power to the board (connected on the VIN pin). The servomotor is also using it as a power supply. The pin on which the servo receives the command with the angle to rotate at is connected to the digital pin 11 of the board, which also supports PWM (can output analog values).

The Bluetooth module connects its TX and RX pins to the digital pins 2 and 3 of the board, which will be configured as RX and TX for the Arduino board. So, the Bluetooth module sends data on its TX pin (digital pin 2 of the board) and will receive data on its RX pin (digital pin 3 of the board) using the SoftwareSerial library.

The KY-028 sensor has two pins we are interested in (apart from GND and VCC). One for digital output and one for the analog output. Although its name states it is a digital sensor, it can output its value in analog format as well. We used the analog output for getting the temperature value, but we connected the digital pin to one of the digital pins of the board also (7, in this case).

Read temperature from sensor and send it over Bluetooth code snippet:

```
if(BT.available() <= 0)
  {
    temp = analogRead(analogPin);
    BT.print(sendDataPackageStartByte);
    BT.print(temp/100);
    BT.print(temp/10%10);
    BT.print(temp%10);
    BT.println();
    delay(communicationDelay);
  }
```

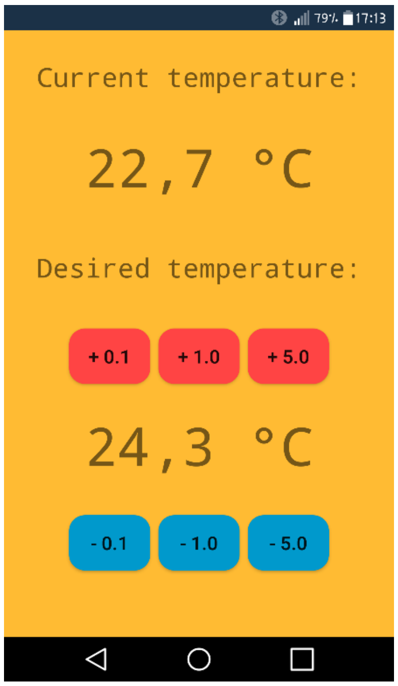
We are reading the analog output of the sensor and divide the value received on digits, to be able to pack it in bytes and send it to the Bluetooth receiver. Some temperature values (for instance 26 ℃, whose received value is 260, cannot be packed inside one single byte, so we have to split it). We also send a start and end of package byte to the receiver, to ensure there are no errors on the transmission.

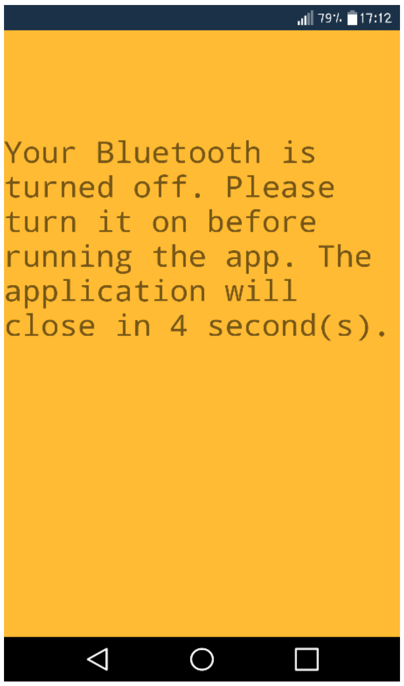
For receiving commands from the user, we follow the same protocol: we wait for data on the channel, we wait to encounter the start byte, we read the digits of the user’s desired temperature, we check the end byte value and then we form the desired temperature based on the read digits. Finally, we map that value (it can range from 150 to 350 - 15 °C to 35 °C) in an angle accepted by the servo – the range 0° – 180°. We also print the received desired temperature to ensure correctness of the transmission.

The code snippet for adjusting the servomotor according to the user command:

```
if (BT.available())
  {
    char temp[3];
    while(BT.available() > 0)
    {
      char crtByte = BT.read();
      if (crtByte == receiveDataPackageStartByteVal)
      {
        temp[0] = BT.read();
        temp[1] = BT.read();
        temp[2] = BT.read();

        if(BT.read() != receiveDataPackageEndByteVal) return;
        break;
      }
    }

    int desiredTemp = atoi(temp);
    int desiredAngle = map(desiredTemp, minTemp, maxTemp, minAngle, maxAngle);
    printDataToSerial(desiredTemp, desiredAngle);

    turnServo(desiredAngle); //attach servo, write value, dettach
  }
```

For the Android part, we will discuss only the Bluetooth module class we developed to ensure Bluetooth communication with the Arduino board (the BluetoothModule.cs class). To establish a connection with the Arduino board, we check the Bluetooth adapter of the device. If there is none, an error message is displayed. We then check if it is turned on and finally if there is any paired device with the specified name. For each case, an appropriate error message is displayed to the user. If everything is fine, we can proceed to read the data the Arduino is sending to our Android device. We get the sent digits, store them inside a buffer, form the current temperature and display it in the user interface. We also check for start and end byte values to ensure transmission correctness. If the user adjusts the temperature using the UI controls, the desired value is sent over Bluetooth (split into digits). The reading and sending of data are performed asynchronously, using the Task functionality provided by C#, which ensures the user interface is not frozen while continuously receiving and eventually sending data back to the Arduino board.





Your Bluetooth is turned off. Please turn it on before running the app. The application will close in 4 second(s).

Read current temperature code snippet:

```
//public async Task<int> ReadCurrentTemp()
await _socket.InputStream.ReadAsync(_buffer, 0, _buffer.Length);
int packageStart = GetPackageStartByteIndex();

if (packageStart == NoDataReceivedCode)
{
    throw new Exception("No data received.");
}
return ComputeCurrentTemp(packageStart);
```

Send desired temperature code snippet:

```
//public async Task SendDesiredTemp(int temp)

byte[] sendBuffer = new byte[PackageLength];
sendBuffer[0] = Encoding.ASCII.GetBytes(SentDataPackageStart)[0];
sendBuffer[1] = Encoding.ASCII.GetBytes((temp / 100).ToString())[0];
sendBuffer[2] = Encoding.ASCII.GetBytes((temp / 10 % 10).ToString())[0];
sendBuffer[3] = Encoding.ASCII.GetBytes((temp % 10).ToString())[0];
sendBuffer[4] = PackageEnd;

await _socket.OutputStream.WriteAsync(sendBuffer, 0, sendBuffer.Length);
await Task.Delay(CommunicationDelay);
```

4. Results

Comparing the initial objectives with the results, we can agree most of the goals were achieved: data is collected and transmitted by the Arduino board, the Android application can connect to the transmitter, receive and also send data back to it. This way, the main goals are reached. The Android application can also intercept and handle different errors (bad transmission, no connection, no device to connect to etc.). Some other goals that could have been achieved were sending additional parameters between the Android and Arduino apps and store it in just one place (the local storage of the Android device, for instance), as some parameters (such as the minimum and maximum allowed temperature) are stored in two places. Also, the user needs to pair to the Arduino board before starting the app. An improvement to this would be to allow him to connect from the app and replace the connection from application startup. But as we previously mentioned, these are only convenience features, the main functionality of the app and the initial goals being reached.

5. Conclusions

The development of this project was quite a challenging, but fun process. I have learnt quite a lot about the Arduino components that I used, but also the software coding side had to win, as I have tried in the past to connect an Arduino board to an Android device, this time managing to do it successfully. Overall, I consider the learnt concepts very useful, as this kind of projects (cross-platform communication, IoT etc.) offer a large variety of possibilities for future projects and developments, such as smart home technology, everyday life gadgets and so on, all of these being made quite easily using platforms like Android and a powerful programming language, such as C#, Java or Python.