Load monitoring and control

(Arduino, Raspberry Pi and IoT)

SESSION 2: DEVELOPING THE COMMUNICATIONS BETWEEN ARDUINO AND RASPBERRY PI

14th/November/2019

Guidelines to communicate and save data from the Arduino

Table of contents

Introduction

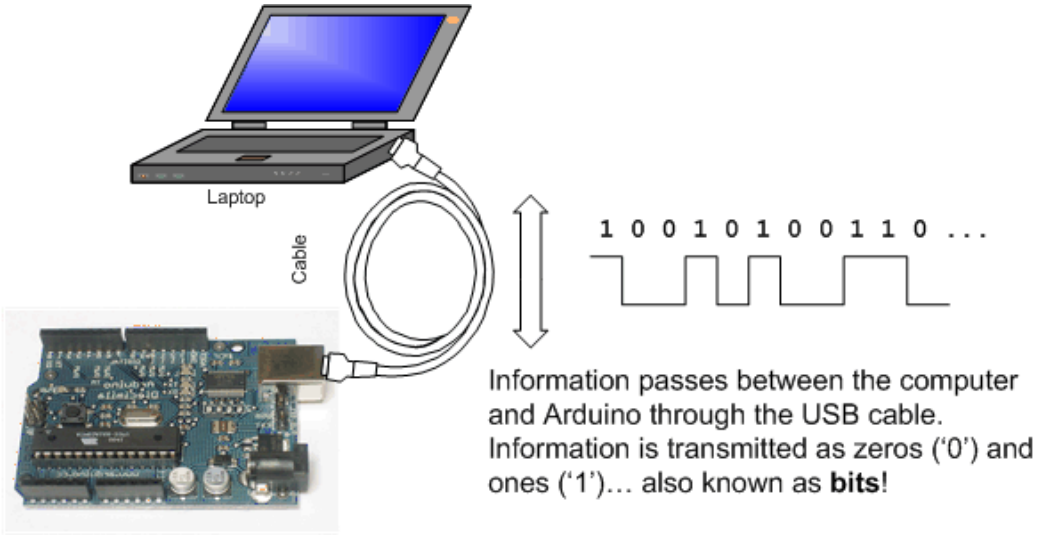
The main goal of the session is to send the data we collect from our Arduino to a computer such as the Raspberry PI or our laptop, and create an output file to store the historic data. Also, we will control the Arduino relay with commands from the computer.

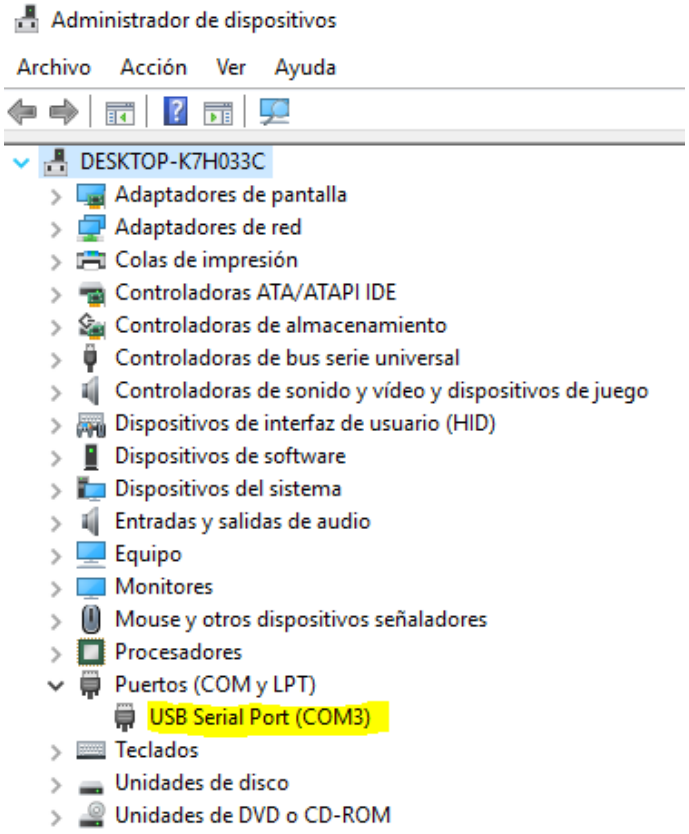
So, today's session is divided in two parts, based on function of the data flow:

- Part 1: The student will send data collected from the Arduino to the computer. In this part the student has to develop an Arduino code for this purpose. Furthermore, it is necessary to collect this data from the computer side, so it will be also necessary to program the computer to read data from the serial port and store it. Later, we will use a Python script to read this data using our Arduino and store it.
- Part 2: In this case the data flow will be the opposite. The computer will send a command to open or close the relay, and the Arduino has to be programmed to read this instruction and control the relay accordingly. In this case, we will do this by means of a Python script.

What is a serial port?

A serial port is an interface that allows a PC to send or receive data one bit at a time. System resources configurations are chosen for each port and are identified by COM1, COM2 and so on. Each COM position represents an input/output (I/O) and an interrupt request (IRQ) address. The I/O address transfers and receives data to and from a peripheral device such as a mouse or keyboard.





Part 1: Sending data from the Arduino to the computer

Part 1a: Collecting and sending data from the Arduino using the Serial Port

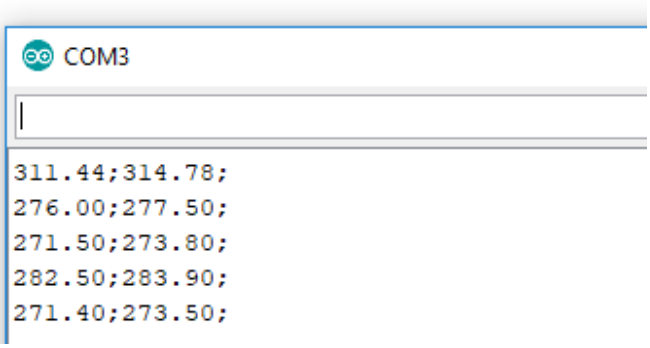
The idea is to store the data you collect from the Arduino. Hence, the main important values to store will be, for example:

- Current (current sensor)
- Temperature (TMP35)
- Humidity (DHT22)
- Light intensity (LDR, phototransistor)
- The relay state

In this session we will work, for example, with temperature sensors and light intensity sensors (LDR). To start with the laboratory session, you have to build up a circuit to collect **two** measurements.

Write the code using the Arduino IDE to get the measurements each 5 seconds and print them in the Serial Monitor.

The result has to be as shown in the next screenshot. Separate the different values with a semicolon. The computer will use the semicolon to identify the order of the received variables.



In this moment we are printing the values every 5 seconds. However, think about how often you want to get data from the current sensor, for example. Do you need a measurement every minute? Is maybe enough to get data every 5 minutes? Even 15?

Part 1b: Connection between the Arduino and your laptop

We are printing the results by means of the serial port. When we use Python, we will use the python script to get the data from the serial port to store them in a csv file, for example. To do this, we need to install and import a new library. To do that **run the anaconda prompt** and write the following command:

```
conda install pyserial

or

conda install -c conda-forge pyserial
```

Further information about the library can be found here:

https://pyserial.readthedocs.io/en/latest/pyserial.html#features

Once we have imported the library, create a new Python script and import the serial library. Later, and when the Arduino code is already collecting and printing data from the connected sensors, we can start writing the python script.

First, we will define from which port we are collecting data from the Arduino IDE. The class that is responsible for that is

```
class serial.Serial
```

__init__(*port=None, baudrate=9600, bytesize=EIGHTBITS, parity=PARITY_NONE, stopbits=STOPBITS_ONE, timeout=None, xonxoff=False, rtscts=False, write_timeout=None, dsrdtr=False, inter_byte_timeout=None, exclusive=None*)

Parameters:

- **port** – Device name or `None`.
- **baudrate** (*int*) – Baud rate such as 9600 or 115200 etc.
- **bytesize** – Number of data bits. Possible values: `FIVEBITS`, `SIXBITS`, `SEVENBITS`, `EIGHTBITS`
- **parity** – Enable parity checking. Possible values: `PARITY_NONE`, `PARITY_EVEN`, `PARITY_ODD` `PARITY_MARK`, `PARITY_SPACE`
- **stopbits** – Number of stop bits. Possible values: `STOPBITS_ONE`, `STOPBITS_ONE_POINT_FIVE`, `STOPBITS_TWO`
- **timeout** (*float*) – Set a read timeout value.
- **xonxoff** (*bool*) – Enable software flow control.
- **rtscts** (*bool*) – Enable hardware (RTS/CTS) flow control.
- **dsrdtr** (*bool*) – Enable hardware (DSR/DTR) flow control.
- **write_timeout** (*float*) – Set a write timeout value.
- **inter_byte_timeout** (*float*) – Inter-character timeout, `None` to disable (default).
- **exclusive** (*bool*) – Set exclusive access mode (POSIX only). A port cannot be opened in exclusive access mode if it is already open in exclusive access mode.

There are several parameters considered in this class. However, we will work only with "port" and baudrate.

- Port → Device name or *none*.
- Baudrate (int) →Baud rate such as 9600 or 115200.

Write a variable name, such as *ser* or *input* for example, and establish the serial port communication using the method:

```
ser = serial.Serial('portname','baurate')
```

Develop a while loop that is always running. Then, read the line from the Serial port. To do this, use the method *readline*()

Once we have read the serial port, we want to print this information when we have new data.

```
while True:
    data = ser.readline()
    if data:
        print(data)
```

Connect your Arduino to your computer and check the configured port serie, for example 'COM3'. Then, run your python script in Spyder IDE. The result has to be something like the shown in the screenshot below:

```
Python 3.7.0 (default, Jun 28 2018, 08:04:48) [MSC v.1912 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.5.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/cristian.chillon/Documents/prova.py',
wdir='C:/Users/cristian.chillon/Documents')
b'309.11;307.11;\r\n'
b'274.20;275.90;\r\n'
b'274.90;276.90;\r\n'
b'284.50;285.50;\r\n'
b'271.80;273.90;\r\n'
b'282.70;283.80;\r\n'
b'272.80;274.70;\r\n'
```

Part 1c. Adding a clock to our data storage script

There is an easy way to include a clock to check the real time within the python script. To do that, import the library datetime. This library is available in Python. Datetime has different objects and datetime classmethods as:

- datetime.now() → return the current local date and time.
- datetime.utcnow() → return the current UTC date and time.

https://docs.python.org/2/library/datetime.html?highlight=now#datetime.datetime.now

Once the date and time are stored, you can consult the data using the next commands:

- year
- month
- day
- hour
- minute
- second

Update your code with the imported library and when the computer receives data from the Arduino, consult the current data and print it together with the received data from Arduino.

```
In [1]: runfile('C:/Users/cristian.chillon/Documents/prova.py',
wdir='C:/Users/cristian.chillon/Documents')
b'303.67;301.56;\r\n'
year: 2018
month: 11
day: 29
hour: 13
year: 20
second: 51
b'277.00;278.30;\r\n'
year: 2018
month: 11
day: 29
hour: 13
year: 21
second: 1
b'277.10;278.70;\r\n'
year: 2018
month: 11
day: 29
hour: 13
year: 21
second: 11
```

Part 1d. Processing the data and storing it in a csv file

At this point, we have all the information from the Arduino and the timestamp corresponding with the CSVs values. However, to do an analysis of the data retrieved with the Smart Plug during the project it is necessary to store de information. So the next step is to save this information in a CSV file.

1) Open a file 'data.csv'
2) If the file doesn't exist, create it.
3) When there is new information from the Arduino, open the CSV file.
4) Add at the end of the file, a new row with the desired values.
5) Close the CSV file.

Part 2: Controlling the Arduino from the computer side

This part is an example of the potential application to control the relay using an algorithm to decide when to close or open the relay of our Smart Plug. For today's session, the algorithm applied will be a simplified one.

Part 2a: Sending a control-signal to the Arduino

The student will have to check the current date and change the relay status every 30 seconds, sending characters as 'H' for HIGH and 'L' for LOW.

Part2b: Reading the Serial Port with the Arduino and follow the signal.

In this part, the student will upgrade the previous Arduino code adding the necessary lines to read information coming from the Serial Port.

It is recommended that this new piece of code to read from the Serial Port goes before the code dedicated to send information. Serial Port is a communication protocol without an established hierarchy for reading and writing. Hence, to prevent collisions between the Arduino and the computer, it is recommended that both the Arduino and the computers/Raspberry Pi check first if there are any pending messages to read.

So, to summarize up:

1) Configure in the setup the relay digital pin to control. If the student doesn't have the Smart Plug, use the LED_BUILTIN that corresponds to the pin number 13

```
pinMode(13, OUTPUT);
```

2) Add an instruction to check if there are any incoming messages at the Serial Port

```
if (Serial.available() > 0) {
}
```

3) If there is a message, store the message in a char variable

```
comando = Serial.read();
```

4) Compare the char variable with the characters 'H' and 'L'.

```
    if (comando == 'H') {
    }
    else if (comando == 'L') {
    }
```

5) If the command was one of the mentioned, change the output pin state.

```
digitalWrite(13, HIGH);
```

or

```
digitalWrite(13, LOW);
```