R package: Arduino UNO Control

User Manual

Gianmarco Polotti

February 3, 2015

1 Abstract

R is a very well known software for statistical computing. However, to make statistic you need data and data often come from experimental measurements. It is quite common to get data from sensors connected to electronic boards. So at the end electronic boards seem in some way a useful tool for statistic. All the previous steps are achieved by using different software to deliver data from the experiments to R. This requires time and additional knowledge before to end up all the process. Recently, the electronic hardware production enjed the "Open Hardware" trend, i.e. the possibility to produce electronic boards with shearable project and design. The trend in electtronic looks very much similar to the trend in software production. In open hardware, one of the most well known examples is the Arduino project [1] and the most well known product is the Arduino UNO board [2].
I think that combining the R and Arduino experiences could be a fruitful idea. If R can directly control the Arduino board then data may come faster and easier to the statistical analysis. With the aim of integrating the R software and Arduino hardware I wrote this package. I used different publications from the web and I tried to cite all of them. In my personal experience, the package is able to send command to the board and get data back from all the Input/Output switches. I am not an electronic expert, so it is evident that much can be done to make the code better. Hopefully, this will stimulate other more experienced people.

2 Package Preface

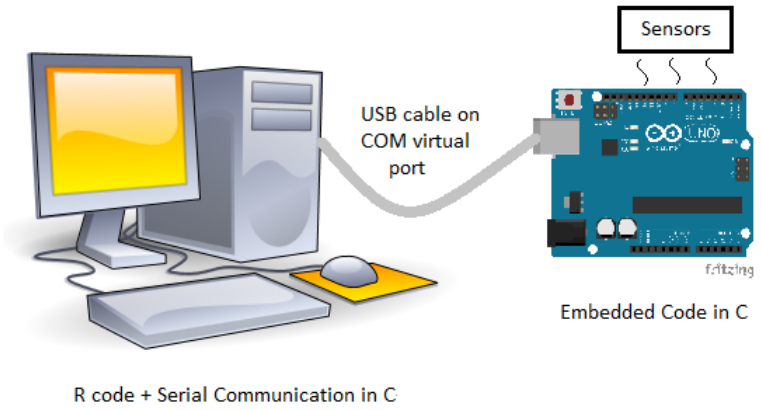
There are previous examples of R and Arduino integration [5], [6], however they use intermediate software (like Processing and Java) to make it possible. This package integrated an adapted library of serial communication in C taken from [7]. The result is a much faster and simpler communication that can be driven by graphical menus.
It is clear that for the use of this package you need the Arduino UNO (original or clone). You can buy one either in a shop or through the internet at very reasonable price, as in the policy of "Open Hardware". After you get your board you need to install the enclosed software [3] (including driver for your operative system) and test it. **Do not go on till you are able to get communication between the board and PC and you are able to load on the board some of the test examples [4].**
After you are sure everything is working, you have to:

1. Take note of the COM port number where the board is connected to (e. g. COM5). This is the communication port used to send commands and get data back. The number is fixed so it does not change unless you move the USB cable in another USB port. The suggested port settings are: 9600 bauds, 8 bits, none parity, 1 bit stop, none flux control.

2. Load on the board the *embedded* part of the package. This software is in "C" and it is necessary because it is able to translate the character string send by R to commands for the different

electronic components of the board. **The embedded software is in the *tool* directory of the package source in the file *driverR.ino* and it is a modified version of [8].** Note that to get it you have to download the source package too. This file is a text file with the C code. The *.ino* extension is typical of Arduino software applications called *Sketch*. The Sketch can be open and load on the micro controller by the Arduino software.

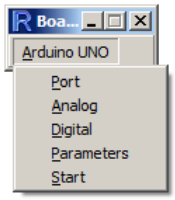
3. On the PC start R and load the ArduinoControl package. A graphic menu tool bar should be automatically loaded.



Please note: the serial communication should work on several Operative Systems as pointed by [7]. However due to the lack of different hardware, I tested only on Windows 7 SO.

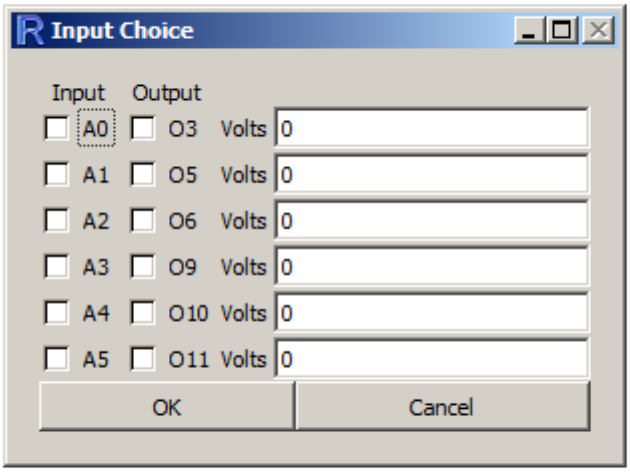
3 Graphical Menus

The graphical tool bar helps the user to make all the steps in the right order. The items in the menu have to be selected one per time from the top to the bottom.

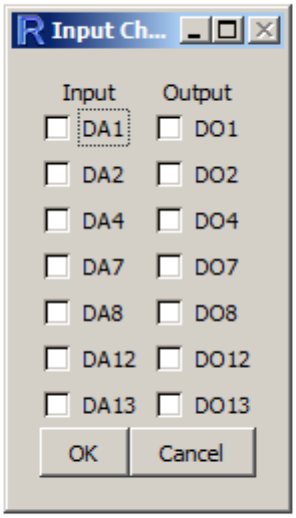


The first choice is for the communication port number. The port number can be easily seen on the Arduino software under the menu : *Tool -> Serial Port*. Be careful to select the right port otherwise any communication can not be established.
The second menu is for Analog IO. Checking the box means that the switch is activated. In case of Input, R will get data from that switch. In case of Output R forces the voltage of the switch from 0 to

the value in the corresponding text box. Consider that only voltage up to 5V may be set.



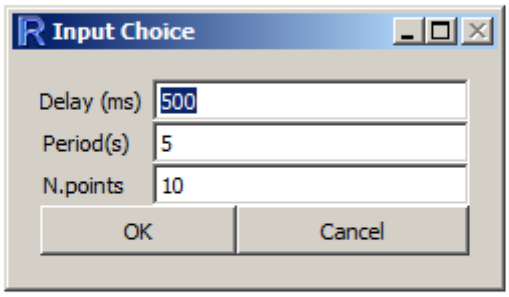
The third menu is for Digital IOs. These are easier to set because the user can only select if the switch should work as Input or Output. In case of Input only Voltage up to 5V are detected and in case of Output the switch is set to 5V constantly.



For both Analog and Digital IOs the switches are labelled exactly as on the board, so the user should not have troubles to identify which IOs are running. After the choice of IOs only few parameters needs to be defined. These are in the fourth menu and they are:

- *Delay* time is an electronic parameter that fixes the interval between two readings. This parameter is key for the board interfacing and it took me sometime to identify it. I suggest not to change it unless the communication does not start at all.
- *Period* is the time between two points plotted on the graph. This is a graphical parameter and must be greater than the previous. If more than one value is collected between two points plotted, the second is the average among all the collected values in the period.
- *N.points* is the number of points that need to be collected. So, the product between this value and the previous one gives the total time (in seconds) of the data acquisition. After the last point is collected the program automatically stops.

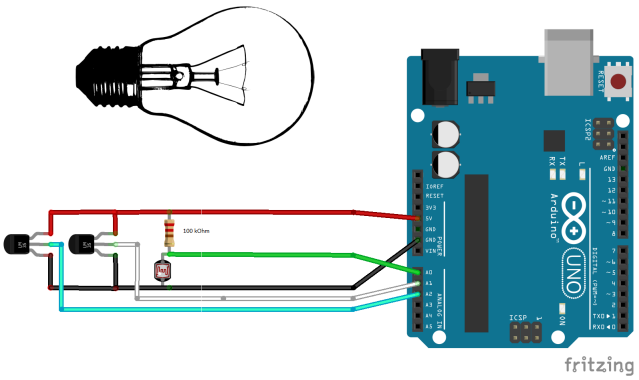
Finally, the last menu starts the data acquisition. Only at this time, R sends to the board an alphanumeric string with the code of the settings. The embedded software decodes the string and acts consequently. If everything goes right a graphical windows is open and the reading of all the inputs are displayed. The graph is updated after each reading and the plot is rightly resized. Readings are the numbers coming from the analog/digital converter of the board so they are integer numbers. A calibration curve needs to be built depending on the meaning of the sensor connected to the correspondent input. The user has to take care of this part. In a future development, it may be interesting to allow a linear calibration of the raw data.
Data collected are stored in the data.frame, *table*, available after the acquisition stops in the object *arduino* on the Global Environment. User can manage the acquired data typing on the console *arduino$table*.

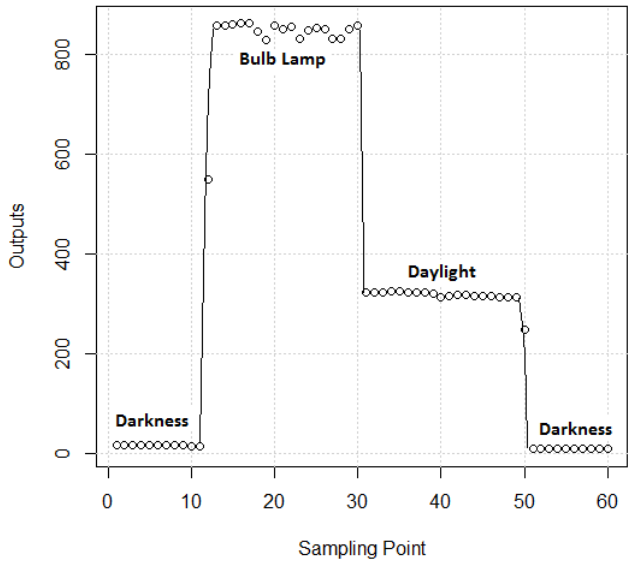


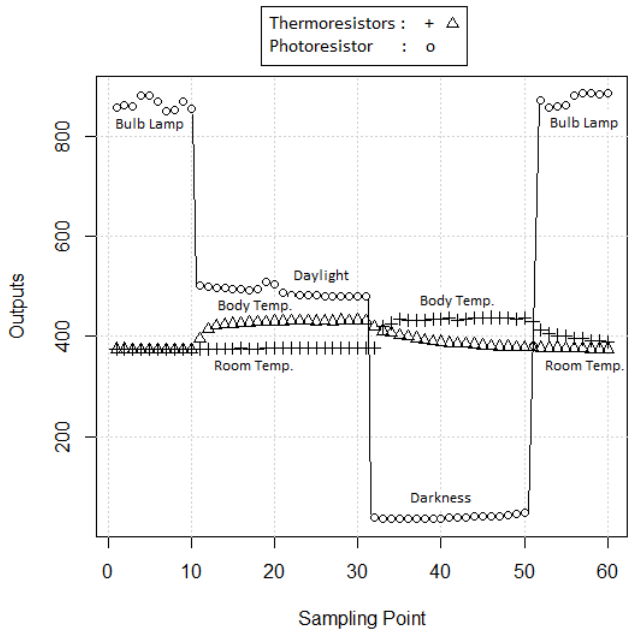
4 Practical Example

Now I show a practical example, but to use the IOs some sensors are needed. Some of them are very simple, cheap and easy to find and connect. I have used a Photoresistor [9] and two Thermoresistors [10]. The first is an electronic component that change its resistance depending of the incoming light. Three levels of light can be easily achieved: the light coming when a bulb lamp is turned on, the daylight of sun and the darkness achived by covering the photoresistor

with a cup. The second components are electric resistances that change with the temperature. Two levels of temperatures are easily accessible: the room temperature and the body temperature if somebody keeps the thermoresistor in his hands.







The previous figures show the basic electrical circuit and the two plots of the incoming readings in case of a single sensors and in case of three sensors. The conditions of acquisition are changed during data acquisition to show how numbers are changing. This example is very trivial but helpful in case of testing hardware and software together.
The use of Arduino and "Open Hardware" in general is becoming common in science thanks to low cost and flexibility of configuration. In the reference I list some of the experiences I have found in literature [11], [12], [13], [14]

References

[1] `http://arduino.cc/`

[2] `http://arduino.cc/en/Main/ArduinoBoardUno`

[3] `http://arduino.cc/en/Main/Software`

[4] `http://arduino.cc/en/Guide/HomePage`

[5] `http://www.magesblog.com/2012/10/connecting-real-world-to-r-with-arduino.html`

[6] `http://jean-robert.github.io/2012/11/11/thermometer-R-using-Arduino-Java.html`

[7] `http://www.teuniz.net/RS-232/`

[8] `http://forums.trossenrobotics.com/tutorials/how-to-diy-128/complete-control-of-an-arduino-via-serial-3300/`

[9] `http://playground.arduino.cc/Learning/PhotoResistor`

[10] `http://playground.arduino.cc/Learning/OneWire`

[11] Qasem Abu Al-Haija, Hasan Al-Qadeeb, Abdulmohsen Al-Lwaimi (2013) *Case Study: Monitoring of Air quality in King Faisal Univaersity using Microcontroller and WSN* The 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2013), Procedia Computer Science 21, 517-521

[12] Sherim Abraham, Xinrong Li (2014) *A Cost Effective Wireless Sensor Network System for Indoor Air Quality Monitoring Applications* The 9th International Conference on Future Networks and Communications, Procedia Computer Science 34, 165-171

[13] Yang B., Patsavas M. C., Byrne R. H., Ma J. 2014 *Seawater pH measurements in the field: A DIY photometer with 0.01 unit pH accuracy*, Marine Chemistry, 160, 75-81

[14] Rosenthal E. 2008 *Make: Liquid ID Spectrometer* `http://creative-technology.net/MAKE.html`