Computer Physics Communications 185 (2014) 2724–2729

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Instrumentino: An open-source modular Python framework for controlling Arduino based experimental instruments☆

Israel Joel Koenka [a,*], Jorge Sáiz [b], Peter C. Hauser [a]

[a] Department of Chemistry, University of Basel, Spitalstrasse 51, 4056 Basel, Switzerland

[b] Department of Analytical Chemistry, Physical Chemistry and Chemical Engineering - University of Alcalá, Ctra. Madrid–Barcelona Km 33.6, Alcalá de Henares 28871, Madrid, Spain

ARTICLE INFO

Article history:
Received 10 April 2014
Accepted 9 June 2014
Available online 18 June 2014

Keywords:
Python
Arduino
Purpose-made instruments
Graphical user interface

ABSTRACT

Instrumentino is an open-source modular graphical user interface framework for controlling Arduino based experimental instruments. It expands the control capability of Arduino by allowing instruments builders to easily create a custom user interface program running on an attached personal computer. It enables the definition of operation sequences and their automated running without user intervention. Acquired experimental data and a usage log are automatically saved on the computer for further processing. The use of the programming language Python also allows easy extension. Complex devices, which are difficult to control using an Arduino, may be integrated as well by incorporating third party application programming interfaces into the Instrumentino framework.

Program summary

Program title: Instrumentino, Controlino

Catalogue identifier: AETJ_v1_0

Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AETJ_v1_0.html

Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland

Licensing provisions: GNU General Public License, version 3

No. of lines in distributed program, including test data, etc.: 17 097

No. of bytes in distributed program, including test data, etc.: 3 425 023

Distribution format: tar.gz

Programming language: Python, C.

Computer: i386, x86-64.

Operating system: Linux, Mac OS X, Windows.

RAM: 60 MB

Classification: 16.4.

External routines: wxPython, pySerial, matplotlib, agw (Instrumentino), SoftwareSerial (Controlino)

Nature of problem:
Control and monitor purpose-made experimental instruments

Solution method:
Modular Graphical User Interface for hardware control

Running time:
Depends on the user.

☆ This paper and its associated computer program are available via the Computer Physics Communication homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author. Tel.: +41 61 267 10 03; fax: +41 61 267 10 13.
E-mail address: yoelk@tx.technion.ac.il (I.J. Koenka).

http://dx.doi.org/10.1016/j.cpc.2014.06.007
0010-4655/

1. Introduction

In the process of scientific research, many laboratories around the world face the need to build their own purpose-made experimental systems. Indeed, this is an inherent feature of scientific research, in which new and unknown phenomena require new experimental set-ups. While some instruments are too complicated to be developed in-house and require professional assistance and ready-made tools from industry, many are simple enough to be realized with limited engineering capabilities available within research groups and from university support staff. In recent years, the scientific community has discovered the open-source electronics platform "Arduino" for monitoring and controlling experimental hardware [1,2]. An Arduino consists of a microcontroller located on a small printed circuit board (PCB) which is fitted with sockets to allow easy connection of external devices to digital and analog input and output (I/O) pins. The success of this particular hardware package stems from the dedicated integrated development environment (IDE), running on a personal computer (PC) under Windows, Mac OS X or Linux, which was designed for the non-expert user and integrates and significantly simplifies the different steps of editing, compiling, and uploading software to the microcontroller. An Arduino can be connected to a separate PCB or a breadboard equipped with interface circuitry to adapt the signals to different components of an experimental system, and thus gain control and monitor abilities. As an open-source project, the Arduino ecosystem offers a significantly cheaper alternative to other hardware control solutions, such as LabVIEW (National Instruments, Austin TX, USA); Arduino boards are available for as little as about 10$.

The popularity of Arduino controllers took off with hobby projects such as wall-avoiding robots and drones [3,4], 3D printers [5], musical instruments [6,7], cellular phones [8], and scientific applications were not late to follow. To date, Arduino controllers have been reported to control various scientific experimental setups. Xoscillo [9], for example, is an Arduino-based oscilloscope and logical analyzer. Do Lago and da Silva [10,11] used an Arduino to control their capacitively coupled contactless conductivity detector (C^4D) for capillary electrophoresis (CE) and high performance liquid chromatography (HPLC). Anzalone et al. [12] reported on an Arduino controlled low-cost colorimeter. Kamogawa et al. [13] have used Arduino to control valves in a flow analysis system. Several Arduino based portable CE instruments were built and successfully used to analyze various samples [14–16]. Several research groups have used the Arduino to monitor ambient conditions such as temperature, humidity and radiation [17–26], as well as pressure and force [19,27–30]. Indeed, many detectors and actuators can be controlled using an Arduino, the possibilities are endless.

The Arduino tool-set provides all the bits and bytes to set up physical control of hardware components. However, it provides only a limited way to interactively control them and there is much room to improve in terms of user interface. The most common and straightforward way to control an Arduino based system is to use the existing Universal Serial Bus (USB), which also powers the Arduino, to exchange information between the Arduino and the PC connected to it. This is easily done using the Arduino *Serial* library and any terminal program on the PC side, but provides only basic capabilities (textual commands and lack of automation and data acquisition). A more sophisticated approach is to write a Graphical User Interface (GUI) that runs on the PC and communicates with the Arduino. Several examples of commercial and free GUIs, built to operate Arduino controlled setups, have been reported. For example, it is possible to create a *MATLAB* (MathWorks, Natick MA, USA) based GUI for Arduino [31] or to use the inherent GUI abilities of *LabVIEW* [32] to control an Arduino. Both of these approaches can produce powerful and customized GUIs for experiments, which are stand-alone or a part of a more complicated control mechanism. Powerful as they are, a clear disadvantage of these approaches is that they require the purchase of expensive commercial products. Open source alternatives exist as well. For example, *DueGUI* [33], offers Arduino GUI libraries for direct connection to a touch screen, instead of an external PC. Another example is *Guino* [34], which employs a fixed GUI environment on a PC, which receives commands from an Arduino. Both of these packages allow the user to create custom GUI with direct control and monitoring capabilities. The problem is that the GUI customization is done in the Arduino sketch and the Arduino microcontrollers have fairly limited resources in terms of memory and processing power. Moreover, such approaches do not easily allow the interaction with other hardware directly connected to the PC, in order to control more complicated instrumentation. A better approach would be to run a small slave program on the Arduino, and control it via a custom-made GUI on a PC. Such slave programs are available [35,36], yet a fully customizable GUI program is still missing.

In this work we present *Instrumentino*, an open-source modular GUI framework, written in Python, to control and monitor Arduino based experimental systems. It communicates with a program running on the Arduino called *Controlino* (a *sketch* in the Arduino world), via a textual master/slave serial protocol (see next sections). One of the advantages is that this sketch is constant between projects and serves only as a mediator between *Instrumentino* on the PC and the physical I/O pins. All of the GUI implementation is done on the PC, and the effort of programming an Arduino is replaced by writing a system description file in Python.

2. Overview

The information flow in an *Instrumentino* controlled experimental system is depicted in Fig. 1. The I/O pins of an Arduino interface different hardware parts via some glue circuitry to properly convert electrical signals, voltage and power levels. The Arduino board is connected via a USB to a PC which is running *Instrumentino* and provides the GUI. Connection to more complex hardware parts can be made via another USB port of the PC (or a legacy interface such as the RS232, a parallel port etc.). These will be controlled by their matching Application Programming Interfaces (APIs), which in turn, are governed by the *Instrumentino* software.

To clarify, a simple example is given in Fig. 2. It consists of an electronic 0–100 psi pressure controller (Parker–Hannifin, Cleveland OH, USA), which is connected to Arduino, and a 3-way valve (LabSmith, Livermore CA, USA), controlled by a dedicated LabSmith controller. The pressure is set with an analog voltage created using the pulse width modulation (PWM) Arduino output pin, and it is monitored with an analog to digital converter (ADC) input channel. The valve is operated using a LabSmith API, integrated into *Instrumentino*.

3. The *Controlino* sketch

The *Controlino* sketch (controlino.ino, see extra material) provides a simple way to control and monitor an Arduino board via the USB cable. It implements the slave side of a textual master/slave protocol (described below), over a 115 200 bps serial connection. On startup, the sketch initializes and begins listening to the serial port for incoming commands. Each command is an ASCII string, terminated by a carriage return (CR) character (ASCII code: 0x0D). Each command starts with a word (*set*, *read*, etc.), followed by a few parameters. The self-explanatory nature of the commands enables a user to directly control the Arduino using any terminal program for debugging purposes. When a CR character is received, *Controlino* parses the received string and acts upon it. When the execution of a command is finished, it replies with a "done!" string, preceded by other relevant data.

Table 1
The available commands in the Controlino sketch. Arguments are given in parenthesis with possible options separated by vertical lines.

set [pin number] [in\out]	Set a digital pin mode to input or output
reset	Set all digital pins to input mode
read [pin1] [pin2]…	Read the values of a list of pins. Pins are given as "A1" or "D1" for analog and digital pins respectively
write [pin number] [digi\anal] [value]	Write a value to a pin (digital or PWM).
SetPwmFreq [pin number] [divider]	Change the PWM frequency of a pin.
softSerConnect [rx pin] [tx pin] [baudrate] [port number]	Initiates a software serial port using a tx pin and an rx pin that enables external interrupts.
hardSerConnect [baudrate] [port]	Initiates a hardware serial connection.
serSend [hard\soft] [port number]	Send a NULL (0x00) terminated string to a serial port.
serReceive [hard\soft] [port number]	Read the RX buffer of a serial port.

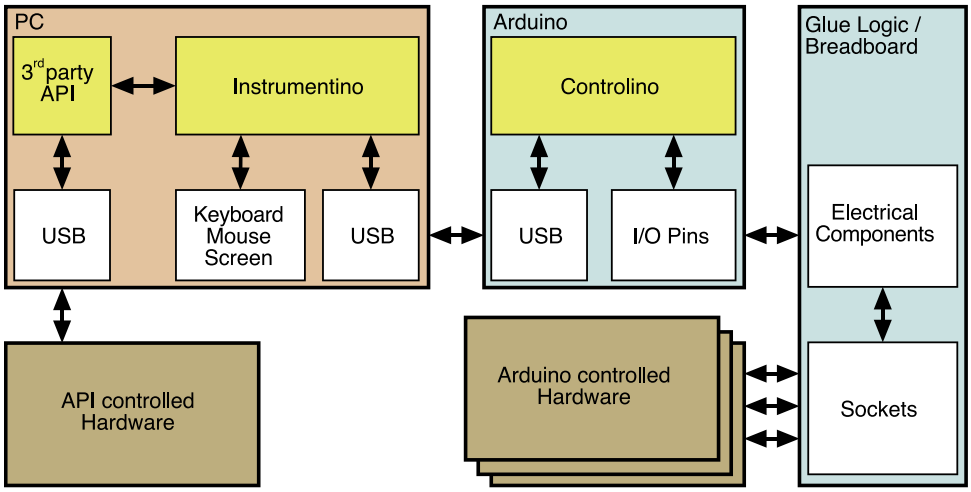

Fig. 1. Data flow in a purpose-made experimental system, using Instrumentino.

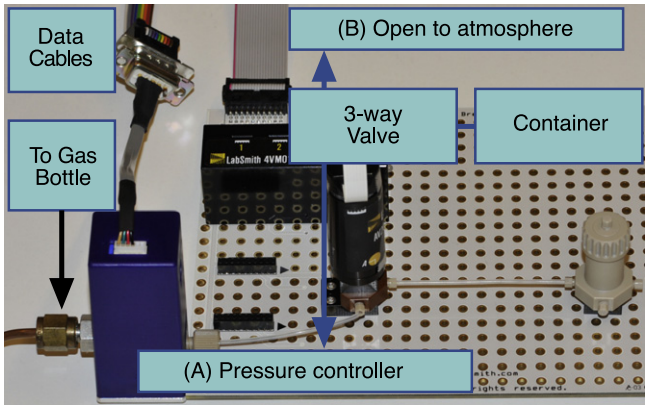

Fig. 2. The hardware components used in the example instrument, overlaid by a corresponding schematic representation.

3.1. The Controlino commands list

The list of available commands implemented in the *Controlino* sketch is shown in Table 1.

Changing the PWM frequency should be done with caution, as it may interfere with other timing related Arduino functions. The number of hardware serial ports is board dependent and up to 4 software serial ports are supported. The *Controlino* sketch is board dependent and different binary files are needed for each type of Arduino board. For now, Arduino boards based on the ATmega328 are supported, as well as the Arduino Mega. Addition of other boards will require adding board specific settings to the sketch (controlino.ino).

4. The *Instrumentino* package

The *Instrumentino* package provides the front-end for communication with the *Controlino* sketch and acts as the master of the aforementioned communication protocol. It allows the user to directly control the desired experimental parameters (such as pressure, and temperature) without the need to be aware of physical control mechanisms (Arduino pins, voltage levels, etc.). The *Instrumentino* framework enables experimenters and system builders to issue user defined commands and complex running sequences to their system, visually monitor system parameters and automatically acquire experimental data for later analysis, all while requiring minimal programming effort. Moreover, *Instrumentino* is built in such a way that allows the concurrent interaction with several hardware controllers, not necessarily only from the Arduino family. This is very useful when some parts of an experimental system are not readily interfaced to an Arduino. Often manufacturers provide devices, which are designed to be connected directly to a PC (*e.g.* via USB) and communicate via an API. In such cases it is much more straightforward to make use of existing code, rather than to attempt to rewrite it for the Arduino platform. Furthermore the hardware and software details of commercial devices may be proprietary and not available. The *Instrumentino* framework is designed to cope with such situations by using available APIs to address pieces of hardware, thus creating a single control program for the entire system.

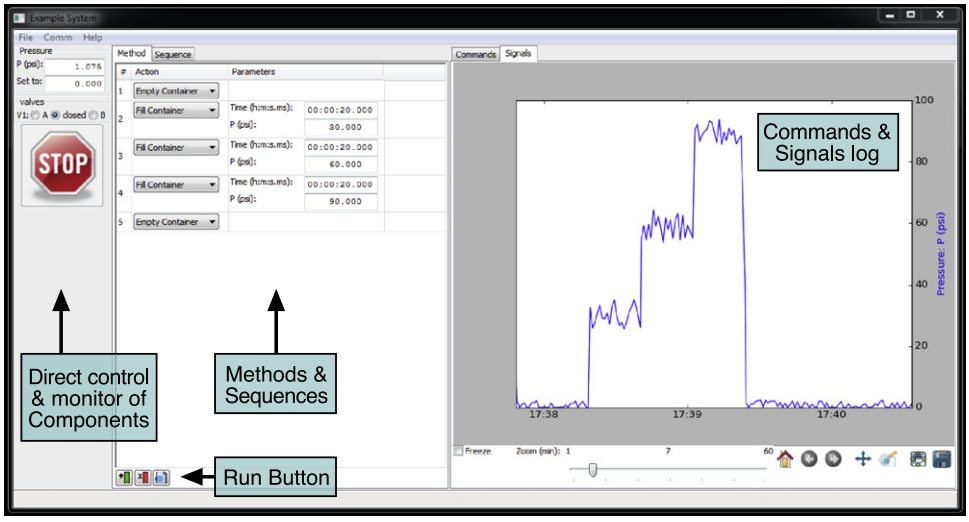

Fig. 3. Screenshot of the Instrumentino application created for container.py. The pressure controller and the valve of the example system can be seen in the components panel. The signal log panel shows how the pressure changed while running the list of actions in the run-control panel.

Python was chosen as programming language for *Instrumentino* because of its ease of use, variety of existing code packages and because it is open source and available for the three operating systems for which the Arduino platform is available. The user interface toolkit chosen for *Instrumentino* was *wxPython*, which is based on the popular cross-platform *wxWidgets* library. It is easy to use and gives a native look on each machine (hence the name wx: Windows & OS X).

To work with *Instrumentino*, the user only needs to provide a system description file (written in Python), which lists the system components and their connections to the Arduino (*e.g.* the Parker pressure controller, connected to the Arduino PWM pin 6 for setting the pressure). It also lets the user define meaningful actions for the system, which can be later executed via the GUI (*e.g.* set the pressure to X psi, wait Y seconds and then turn it off, X & Y being parameters). By listing the components and the actions of a system, *Instrumentino* is provided with all information required to create the specific GUI.

4.1. The system description file structure

The description file is logically divided into the following sections (an example is given in the supplementary material (see Appendix A), container.py):

- **Imports**
  As in any Python module, import statements declare classes and objects defined in other source files. By importing them, it is possible to reference them in the module.
- **Constants**
  It is good coding practice to assign meaningful names to constants relevant to the system, such as Arduino pin assignments.
- **Components**
  The instantiation of objects for each component in the experimental system. Each system component is described in the code by a Python class, which exists in the *Instrumentino* package. Hardware components currently supported include high voltage power supplies, pressure and mass flow controllers, syringe pumps, two-, three- and multi-position valves, and data recording systems.
- **Actions**
  The declaration of basic actions to be performed by the system. Each action is defined as a class, which inherits from the Python class *Instrumentino.action.SysAction*, and needs to implement a method named *Command()*. This method will be called when the action is to be executed.
- **System definition**
  Wrapping up all of the above and some more information (such as the system name and description) in a class, which inherits from the *Instrumentino.Instrument* class.
- **Program run**
  The commands to be executed when this Python file runs. To run the application one needs simply to instantiate the System class defined in the last section. Typing "python container.py" in a terminal will start the application.

4.2. The graphical user interface

The GUI, described by container.py, can be seen in Fig. 3. It features three panels: *components*, *run-control* and *log*, appearing from left to right. The components panel allows the user to directly control and monitor the system components. Each component is represented by a sub-panel and is created according to the components list in the system description file. A component panel is composed of its associated digital and analog variables. A digital variable in a system might be the different positions of a valve (e.g. *A*, *B* or *closed* for the LabSmith valve in the example) or a trigger signal (*on* or *off*). A digital variable is represented by a set of radio buttons with the possible options. An analog variable might be a measurement of some physical quantity (*e.g.* pressure) and is represented by a text box for reading the value and an optional text box for setting it, within the allowed range. The components panel also features a large stop button, to be used in case an experiment needs to be shut off abruptly.

The run-control panel has two modes. The first operates with methods. A method is a successive list of actions, from the actions pool defined in the description file. It conceptually describes a way to achieve a specific experimental goal. The action in each line in the list is picked using a combo-box, and its relevant parameters can be set. The user can then run the method using the run button or save the newly defined method as a *.mtd* file using the *File* menu. The second mode manages sequences, in which the user can create a sequence, composed of previously saved methods. Each method can be run a number of consecutive times, so a whole day of experiments or a long-term monitoring experiment can be planned and run automatically. A defined sequence can also be saved, as a *.seq* file, using the *File* menu.

The *log* panel to the right has two modes as well. One is the *command-log*, which logs each run of a method or a sequence. The

contents of this list are automatically saved as a *.txt* file using the date and time as the file name. It acts as a lab journal and can be referenced later to report on the exact experimental conditions. Another mode is the *signal-log*, an interactive timeline graph, which shows the physical parameters measured by the system. Each parameter has its own *y* axis and a different color on the graph. This data is automatically saved as well as a *.csv* file (comma separated values) using the same file name as the *command log*. The file can be later opened in spreadsheet programs such as Excel (Microsoft Office) or the open source Calc (LibreOffice) for further data analysis.

4.3. Program workflow

When the application starts, the panels are disabled. It is only possible to create methods and sequences until all the required controllers are connected, using the *Comm* menu. Once communication is established, system variables (pressure and valve position in the example) are periodically read and displayed in the left panel, and in the signal log panel. The user can then run a method and observe the signal log. The screenshot in Fig. 3 was taken after a method had been created and run (see center panel). It set the valve to port A to fill the container with gas and set the pressure inside to 30, 60 and 90 psi in steps of 20 s. Afterwards it opened the container by setting the valve to port B. The resulting pressure readings can be seen in the log panel.

The *Instrumentino* code package is available as a Python egg which encapsulates both the Python code and the resource files needed for operation. It can be downloaded from the Computer Physics Communications Program Library (http://cpc.cs.qub.ac.uk/summaries/AETJ_v1_0.html), PyPI repository (https://pypi.python.org/pypi/instrumentino) or Github (https://github.com/yoelk/Instrumentino). Users of *Instrumentino* will need to install the egg file on their computer, create a Python module to describe their experimental set-up and run it (similar to container.py). It is released under the GPLv3 license, to ensure its open-source nature in the future.

5. Conclusions

In research, the construction of purpose made experimental systems is frequently necessary. The introduction of open source electronics, in particular the Arduino platform, is an enabling factor, in providing researchers a tool to build relatively complex systems, which were previously only possible with significantly more effort or were beyond reach. The *Instrumentino*/*Controlino* software package was developed to streamline the development of the control software for such systems and to allow full automation in order to improve performance or to enable unattended operation. It was built in such a way that only a minimal programming effort is required, greatly simplifying the task of instrument control and making it accessible for less experienced programmers. Since its development, *Instrumentino* has been used extensively in our research groups, saving a significant amount of time and effort, and we believe it can assist other researchers as well in achieving their goals. As a free open source project, users are welcome to contribute and extend *Instrumentino* to support more kinds of controllers and hardware components or to add features they find useful. There is much potential to grow and the use of *Instrumentino* is, of course, not limited to scientific purposes.

Acknowledgment

The authors are grateful for financial support by the Swiss National Science Foundation through grants 200020-137676 and 200020-149068.

Conflicts of interest

The authors have declared no conflict of interest.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.cpc.2014.06.007.

References

[1] J.M. Pearce, Building research equipment with free, open-source hardware, Science 337 (2012) 1303.
[2] A. D'Ausilio, Arduino: a low-cost multipurpose lab equipment, Behav. Res. Methods 44 (2012) 305.
[3] Brandon121233, Wall avoiding Robot, http://www.instructables.com/id/Make-a-wall-avoiding-Robot!/. Retrieved April 2014.
[4] APM, Ardupilot—official website, http://ardupilot.com/. Retrieved April 2014.
[5] A. Bowyer, RepRap—official website, http://reprap.org/wiki/RepRap. Retrieved April 2014.
[6] J.H. Owen Vallis, Arduinome—official website, http://flipmu.com/work/arduinome/. Retrieved.
[7] Kylemcdonald, Arduino based guitar pedal, http://www.instructables.com/id/Lo-fi-Arduino-Guitar-Pedal/. Retrieved April 2014.
[8] Xiaobo, ArduinoPhone, http://www.instructables.com/id/ArduinoPhone/. Retrieved April 2014.
[9] Aguaviva, Xoscillo, http://code.google.com/p/xoscillo/. Retrieved April 2014.
[10] J.A. Fracassi da Silva, C.L. do Lago, An oscillometric detector for capillary electrophoresis, Anal. Chem. 70 (1998) 4339 (1998/10/01).
[11] K.J.M.F. Claudimir Lucio do Lago, OpenC4D—official website, https://sites.google.com/site/openc4d/home. Retrieved April 2014.
[12] G.C. Anzalone, A.G. Glover, J.M. Pearce, Open-source colorimeter, Sensors 13 (2013) 5338.
[13] M.Y. Kamogawa, J.C. Miranda, Use of "arduino" open source hardware for solenoid device actuation in flow analysis systems, Quimica Nova 36 (2013) 1232.
[14] T.D. Mai, et al., Portable capillary electrophoresis instrument with automated injector and contactless conductivity detection, Anal. Chem. 85 (2013) 2333. 2013/02/19.
[15] J. Sáiz, T.D. Mai, P.C. Hauser, C. García-Ruiz, Determination of nitrogen mustard degradation products in water samples using a portable capillary electrophoresis instrument, Electrophoresis 34 (2013) 2078.
[16] J. Sáiz, et al., Rapid determination of scopolamine in evidence of recreational and predatory use, Science & Justice 53 (2013) 409.
[17] G. Gasparesc, Development of a low-cost system for temperature monitoring, in: 2013 36th International Conference on Telecommunications and Signal Processing, TSP, 2013, p. 340.
[18] N. Barroca, et al., Wireless sensor networks for temperature and humidity monitoring within concrete structures, Constr. Build. Mater. 40 (2013) 1156.
[19] M.R. Dehmlow, Asme, Affordable universal solar tracker, in: Proceedings of the Asme 5th International Conference on Energy Sustainability 2011, Pts a-C, 2012, pp. 653–661.
[20] A. Abdullah, et al. Development of wireless sensor network for monitoring global warming, in: 2012 International Conference on Advanced Computer Science and Information Systems, 2012, pp. 107–111.
[21] S.-h. Sun, Y.-s. Jin, W.-j. Zhang, Asme, Design and simulation of remote temperature monitor and control system based on embedded web server, in: 2011 International Conference on Instrumentation, Measurement, Circuits and Systems, 2011, pp. 297–300.
[22] M.G. Rodriguez, et al. Wireless sensor network for data-center environmental monitoring, in: 2011 Fifth International Conference on Sensing Technology, ICST 2011, 2011, p. 533.
[23] A.H. Kioumars, T. Liqiong, Wireless network for health monitoring: heart rate and temperature sensor, in: 2011 Fifth International Conference on Sensing Technology, ICST 2011, 2011, p. 362.
[24] J.M. Gomes, P.M. Ferreira, A.E. Ruano, Implementation of an intelligent sensor for measurement and prediction of solar radiation and atmospheric temperature, in: 2011 IEEE 7th International Symposium on Intelligent Signal Processing, WISP 2011, 2011, 6 pp.
[25] F.J. Garcia-Diego, J.J. Pascual, F. Marco-Jimenez, Technical note: design of a large variable temperature chamber for heat stress studies in rabbits, World Rabbit Sci. 19 (2011) 225.
[26] R. Gomaa, I. Adly, K. Sharshar, A. Safwat, H. Ragai, ZigBee wireless sensor network for radiation monitoring at nuclear facilities, in: Proceedings of 2013 6th Joint IFIP Wireless and Mobile Networking Conference, WMNC 2013, 2013, 4 pp.
[27] M. Stalin, C.L. Bennett, in: R. Jung, A.J. McGoron, J. Riera (Eds.), 29th Southern Biomedical Engineering Conference, 2013, pp. 137–138.
[28] F. Grenez, M. Viqueira Villarejo, B. Garcia Zapirain, A. Mendez Zorrilla, Wireless prototype based on pressure and bending sensors for measuring gate quality, Sensors (Basel, Switzerland) 13 (2013) 9679.

[29] L. Russell, A.L. Steele, R. Goubran, Ieee, in: 2012 Ieee International Instrumentation and Measurement Technology Conference, 2012, pp. 2695–2699.
[30] R.P. Rush, Acm, sensation augmentation to relieve pressure sore formation in wheelchair users, in: Assets'09: Proceedings of the 11th International Acm Sigaccess Conference on Computers and Accessibility, 2009, pp. 275–276.
[31] Rmagtibay, Arduino I/O-Matlab basic tutorial, http://www.instructables.com/id/Arduino-IO-Matlab-basic-tutorial/. Retrieved April 2014.
[32] National Instruments, NI LabVIEW Interface for Arduino Toolkit http://sine.ni.com/nips/cds/view/p/lang/en/nid/209835. Retrieved April 2014.
[33] Cowasaki, DueGUI, http://www.cowasaki.co.uk/DueGUI/DueGUI_0_13.pdf. Retrieved April 2014.
[34] Madshobye, Guino: Dashboard for your Arduino, http://www.instructables.com/id/Guino-Dashboard-for-your-Arduino/. Retrieved April 2014.
[35] Arduino, Arduino and Python, http://playground.arduino.cc/interfacing/python-.UxY0Tl7hO2y. Retrieved April 2014.
[36] Intructable, Control your arduino from your PC with the Qt Gui, http://www.instructables.com/id/Control-your-arduino-from-your-PC-with-the-Qt-Gui/. Retrieved April 2014.