Available online at www.sciencedirect.com

ScienceDirect

IFAC-PapersOnLine 48-29 (2015) 158–163

Remote Web-based Control Laboratory for Mobile Devices based on EJsS, Raspberry Pi and Node.js

J. Bermúdez-Ortega [*] E. Besada-Portas [*] J.A. López-Orozco [*] J.A. Bonache-Seco [*] J.M. de la Cruz [*]

[*] *Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Madrid, Spain (e-mails: juliberm@ucm.es, evabes@dacya.ucm.es, jalo@dacya.ucm.es, jabonache@ucm.es, jmcruz@ucm.es)*

Abstract: This paper presents a new approach to develop remote practices for Systems Engineering and Automatic Control laboratories based on Easy JavaScript Simulations (EJsS), Raspberry Pi and Node.js. EJsS is used to create a JavaScript and HTML5 laboratory front-end that lets teachers and students parametrize and observe the behavior of the controllers/systems under study from the web-browsers of their tablets and smartphones. The Raspberry Pi, a low-cost single-board computer, is in charge of running 1) the C program that closes the control loop over the selected plant and 2) the JavaScript laboratory web server that hosts the laboratory front-end and communicates it with the controller. Finally, the JavaScript development and runtime platform Node.js is used 1) to develop and run the Javascript laboratory web sever within the Raspberry Pi and 2) to support the communications between the laboratory front-end and server. The new strategy, tested over a Proportional/Integral/Differential (PID) controller for a vertical mono-rotor plant, provides low-cost real-time support to the controller and friendly remote access from mobile devices to the practices for the students.

Keywords: Virtual and Remote Labs; Virtual Reality; Internet-Based Control Education; mobile devices; EJsS; Node.js; Raspberry Pi.

1. INTRODUCTION

Remote web-based laboratories are learning/teaching resources in those scientific and technical disciplines where 1) the observation of the actual behavior/response of physical phenomena is crucial, 2) the costs or difficulties to assemble the experimental setup are meaningful, and 3) the time and equipment restrictions associated to tradicional hand-on laboratories want to be overcome (Alhalabi et al. (2000); Sivakumar et al. (2005); Chang et al. (2005); Salzmann and Gillet (2007)).

They are becoming successful e-learning tools as they let students 1) interact with the system changing parameters, 2) observe the results of their manipulations, 3) measure real data, 4) perform an almost unconstrained amount of experiments, and 5) study phenomena which would not be possible to investigate in a traditional (hands-on) laboratory. Some examples within the Control Engineering field are the control of the water level in a multi-tank system (Grau and Bolea (2008); Dormido et al. (2008); Stefanovic et al. (2011)), the control of a heat exchanger (Lazar and Carari (2008)), the position/velocity control of a servo motor (Vargas et al. (2008)), the attitude control of a fix-located quadrotor (Besada-Portas et al. (2013a)) and the control of mobile robots (Chaos et al. (2013)).

[⋆] This work has been supported by the Spanish National Research Project DPI2013-46665-C2-1-R.

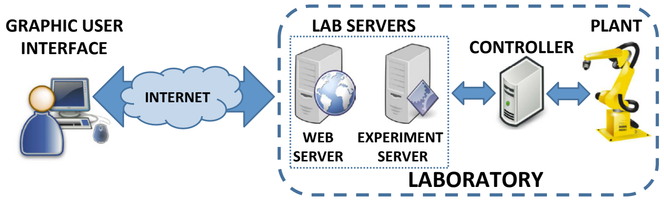


Fig. 1. Schema of a generic web-based laboratory

A common approach in web-based laboratories for Control Engineering is to implement them using the following elements (whose relationships are represented in Fig. 1): a Graphical User Interface (GUI) that lets the students parametrize the behavior of the controller and observe the evolution of the system/controller signals, a controller application that is in charge of closing the feedback loop over the plant under study, and the servers that handle the access to, and the connection between, the GUI and the controller.

Although the software/hardware used to implement/deploy each element differs from one lab to other, some tools are gaining space within the educational community. For instance, the laboratory front-ends (i.e. its GUIs) are often implemented as Easy Java Simulations (EJS) applets to be able to run them directly from the laboratory webpage. Its back-end (i.e. the controller) sometimes takes advantage of the capabilities of Matlab (Farias et al. (2008); Fabregas

2405-8963 © 2015, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.
Peer review under responsibility of International Federation of Automatic Control.
10.1016/j.ifacol.2015.11.230

et al. (2011)), while others benefits from the robustness of LabVIEW (Dormido et al. (2008); Vargas et al. (2008); de la Torre et al. (2011); Chacon et al. (2015)) or TwinCAT (Besada-Portas et al. (2012, 2013b)). Finally, the complexity of the lab servers varies from basic applications that connect the GUI and the controller (Besada-Portas et al. (2012)) to several applications for hosting the laboratory webpage, connecting the GUI and the controller, and handling collaborative/exclusive access to the laboratory resources (de la Torre et al. (2013, 2015)). All are successfully used in different courses at the universities where they were developed and some are integrated in a network[1] that shares laboratory resources among universities.

However, the technological advances in mobile devices and low-cost single-board computers, and the current state of Easy JavaScript Simulations (EJsS[2]) and Node.js[3] offer new possibilities for the development of web-based control laboratories that preserve many of the benefits of the existing ones while solving some of their current drawbacks, such as the impossibility to use Java applets in webpages within iOs/Android/WindowsMobile-based mobile devices, or the costs associated to the Matlab/LabVIEW/TwinCAT licenses and the laboratory deployment computers.

This paper presents our approach to develop new low-cost web-based laboratories with a friendly GUI embeddable in webpages that can be used from the students mobile devices (smartphones or tablets) and computers. In short, our strategy consists in using EJsS to obtain a portable laboratory front-end, Node.js to develop/run the server that connects the laboratory GUI with the controller application implemented in C, and the low-cost single-board computer Raspberry Pi to deploy the server and controller.

The rest of the paper is organized as follows. Section 2 presents the technology (EJsS, Node.js and Raspberry Pi) and the elements (GUI, server and controller applications) that are used in our remote web-based control laboratory. Section 3 presents a Control Engineering experiment prepared using the selected tools and methodology. Finally, Section 4 contains our conclusions regarding this work.

2. LABORATORY ELEMENTS

This section explains the characteristics of the hardware and software elements of our new remote control laboratory, schematized in Fig. 2, and highlights their most relevant features. On one hand, the laboratory front-end, presented in Section 2.4, is 1) implemented as a JavaScript and HTML5 webpage using EJsS and 2) accessed from the students mobile devices or personal computers. On the other one, its server and controller, described in Sections 2.3 and 2.2, are 1) developed with Node.js and C, and 2) deployed in the Raspberry Pi introduced in Section 2.1.

2.1 Raspberry Pi: Server & Controller Hosting

The Raspberry Pi is a low-cost small-size single-board computer originally developed in the United Kingdom between 2006/2012 with the purpose of promoting Computer Science in elementary and middle school students. However, its low cost (35US$), credit-card size and connectivity possibilities, as well as the information provided by many of its users through different blogs and webpages[4], have made it a reference embedded system in many technical university studies and in some industrial applications.

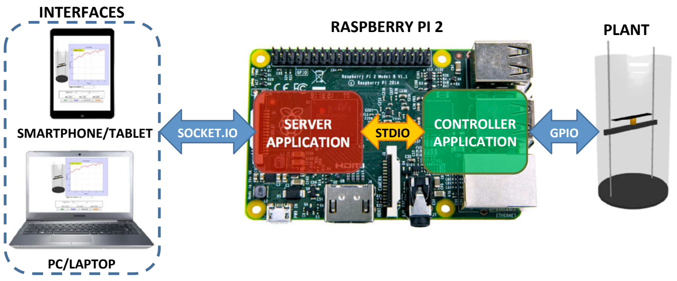


Fig. 2. Schema of the new web-based laboratory

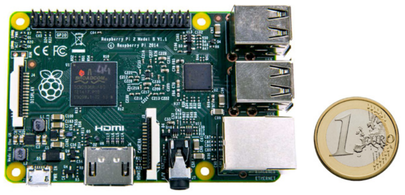


Fig. 3. Raspberry Pi 2

Its most recent version, the Raspberry Pi 2 (see Fig. 3) has a quad-core ARM Cortex-A7 CPU, 1GB RAM, 4 USB ports, 1 Ethernet connexion and multiple General Purpose Input/Output (GPIO) pins (plus some others configurable to provide serial I2C, SPI and RS-232 connectivity). These characteristics make it ideal to deploy a light-weighted server for the laboratory and a controller application, which interacts with the sensors/actuators of the real plant through the Raspberry external connections.

The Operating System (OS) used in the our new laboratory web-based strategy is Raspbian[5], a robust precompiled Debian especially optimized for the Raspberry Pi which is bundled with, or can be extended with the installation of, many software packages that have already been tested for a big community of users. Among the packages to be installed, the controller application requires the WiringPi, and the laboratory server needs a Raspbian version of Node.js and socket.io[6]. Finally, we want to highlight the free-cost of the software elements (Raspbian, WiringPi, Node.js and socket.io) used in the Raspberry Pi, an essential feature to maintain the low-cost of our lab.

2.2 Controller Application

The controller application, deployed in the Raspberry Pi, is in charge of closing the measurement-control-action loop over the selected plant and of exchanging information with the laboratory GUI through the laboratory server.

To perform both tasks, we implement it, as Fig. 4 schematizes, as a C application with two threads. On one hand,

[1] UNILabs: http://unilabs.dia.uned.es/
[2] EJsS: http://www.um.es/fem/EjsWiki/pmwiki.php
[3] Node.js: https://nodejs.org/industry/
[4] Raspberry Pi: https://www.raspberrypi.org/blog/
[5] Raspbian: https://www.raspbian.org/
[6] See Sections 2.2 and 2.3 for a description of the utility of these software packages.

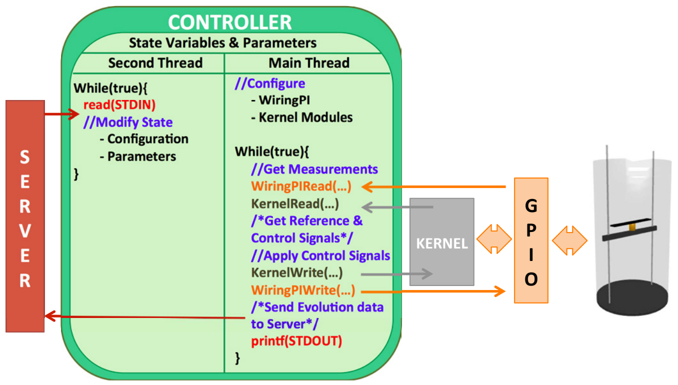


Fig. 4. Controller application schematized behavior

the main thread periodically closes the control loop over the plant and sends to the server the values of those signals that are represented in the evolution curves of the GUI to let the students analyze the system/controller behavior. On the other, the second thread receives/interprets the configuration/parametrization messages generated by the students interaction with the GUI that are sent through the server to the controller.

The communication of the controller application and the sensors/actuators used in the experiments can take advantage of the capabilities of the WiringPi[7]: a software library that facilitates the configuration/access of the Raspberry Pi GPIO pins from a C-based application. Alternatively, the interaction with the hardware can be supported by kernel modules, to be developed for certain sensors/actuators that require a special handling not supported by, or with better performance that the provided by, WiringPi. This additional modules will deliver/receive the information through Character Device Files[8] (CDFs) that will be accessed from the controller application. The experimental setup presented in Section 3 will illustrate the benefits of using both ways of controller-hardware communications.

Finally, the controller application receives/sends the configuration&parametrization/evolution messages using the application STandard INput/OUTput (STDIN/STDOUT). Therefore, it uses the C read/printf routines to exchange the information with the server. All the messages are strings of characters with a starting label that lets the controller application identify its meaning and a chain of values separable with the ':' token.

2.3 The Node.js-based Laboratory Server

The laboratory server, deployed in the Raspberry Pi, is in charge of 1) delivering the laboratory webpage that hosts the GUI of our experiments, 2) opening the connections with the lab GUIs and controllers, 3) exchanging the data (text messages) between the lab GUIs and controllers and 4) launching the controller application when it is not already running and there is a GUI trying to establish a connection with the laboratory back-end.

Our server is implemented with Node.js, an open-source multi-platform development and runtime environment for

[7] WiringPi: http://wiringpi.com/
[8] The linux kernel module programming guide by P. J. Salzman, M. Burian and Ori Pomerantz: http://tldp. org/LDP/lkmpg/2.4/html

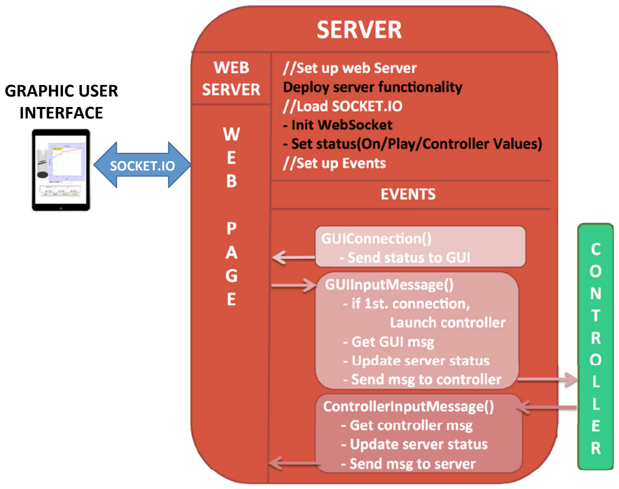


Fig. 5. Laboratory server schematized behavior

scalable and fast network applications, especially intended to support JavaScript-based web servers. Its asynchronous event-driven non-blocking I/O architecture makes it a good choice for data-intensive real-time applications, distributed over different devices, such as our laboratory. Its underlaying Google V8 JavaScript engine[9] permits us to develop a computationally lightweight and efficient laboratory server which can be run in different OS such as OS X, Microsoft Windows or Linux (including Raspbian). Its big community of users, which includes firms such as LinkedIn, eBay or Microsoft[10], makes it a previously well-tested platform to develop/run the server of our laboratory. Last but not least, it lets us use the same programming language, JavaScript, both in the laboratory GUI and server.

The communication between the laboratory GUI and server is supported by socket.io[11], one of the modules that expands the functionalities of Node.js[12], that provides, when it is possible, a full-duplex WebSocket communication between two JavaScript applications (our EJsS GUI and Node.js server). This type of communication is handled by the event-driven Node.js philosophy, and therefore, permits our lab server to efficiently redirect the messages received from the GUI to the controller and viceversa.

The behavior/steps of the JavaScript file that implements our server is/are schematized in Fig. 5. It starts setting up a web server that, when accessed through Internet, will send the HTML5 webfile that implements, using the JavaScript commands generated by EJsS, our laboratory GUI. Next, it loads the socket.io module, and initializes a WebSocket and the variable that determines the status of the laboratory (on/play/controller values). Then, it sets up the events (GUI connection, GUI input message and controller input message) and their functionality, which are in charge of driving the behavior of the server. At GUI connection, it sends the data status to the GUI. At GUI in-

[9] This engine, especially designed for Chrome, compiles the JavaScript code into machine code before its execution and optimizes it during the running time.
[10] Node.js industrial users: https://nodejs.org/industry/
[11] Socket.io: http://socket.io/
[12] Other modules developed for Node.js provide it with additional features such us file management, networking (HTTP,TCP), cryptography, etc.

put message, it receives a message through the WebSocket that connects the GUI with the server, updates the status of the server and redirects the message to the controller through its STDIN. Additionally, when it receives the first GUI input message, it uses the Node.js child_process[13] module to launch the controller application as a child process of the server and to automatically open a communication thread with the STDIN/OUT of the controller. Finally, at controller input message, it receives the message from the controller STDOUT, updates the status of the server and redirects the message to the GUI. In short, it sets up the webpage, launches the controller, establishes the connections with the laboratory front&back-ends and redirects the asynchronous messages that are received from one side of the laboratory to the other.

2.4 The EJsS-based Laboratory GUI

Our laboratory GUI is the tool used by the students from their mobile devices or computers to access the experiences of our remote laboratory, configure/parametrize the behavior of the controllers, and observe the evolution of those variables that are of interest in the analysis of the system/controller behavior.

It is implemented with Easy JavaScript Simulations, a freeware, open-source tool developed in Java to help users with low programming skills to create discrete computer simulations in a simple, graphical and intuitive way, following the Model-View-Control paradigm (Esquembre (2004)).

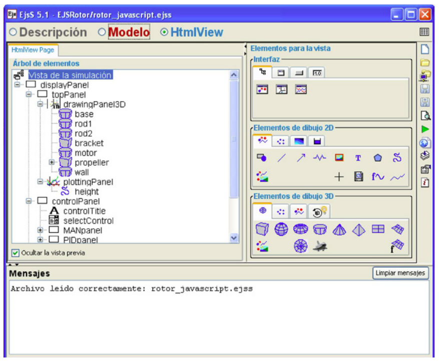
Besides its simulation-development possibilities, it is a powerful tool to develop a graphical/interactive interface, because its visual capabilities, originally designed to display the simulations, permit any user to create GUIs with different types of displays and control elements. For this reason, the original version of the tool, Easy *Java* Simulations (EJS), has been adopted to develop the Java applet GUIs of several remote laboratories, which can be accessed from the web browsers of the students personal computers but not from their mobile devices[14].

The new version of the tool, available from January 2014 and called Easy *JavaScript* Simulations (EJsS), follows the same Model-View-Control paradigm than the previous and preserves many of its displaying and interactivity capabilities (which users familiar with EJS can recognize in Fig. 6). However, it converts the simulation into a HTML5 + JavaScript webpage that can be called, due to the JavaScript encoding of the simulation, from the web browsers available in mobile devices and computers[15]. Therefore, it seems a good choice to develop the front-ends of new remote laboratories, as the students will be able to access the experiments from the mobile devices that they carry everywhere and from their laptops/home computers.

[13] Child_process:https://nodejs.org/api/child_process.html

[14] In order to run a JAVA applet from a web browser, personal computers have a plugin installed which is not available in the usual types of mobile devices (such us those based on Android, iOs or Windows Mobile).

[15] It also permits to generate simulation apps for Android and iOs mobile devices that can be run from the EJsS Android and iOs Reader App, although this characteristic is not exploited in our current remote laboratory approach.



```
Archivo leido correctamente: rotor_javascript.ejss
```

Fig. 6. EJsS programming interface

To prepare the GUI of the lab within EJsS, we define 1) its view through the HtmlView panel and 2) the variables/functionality used to build the messages that are sent to the server when the student interacts with control elements (buttons, numeric fields, selectors, ...) of the GUI through the Model panel. Next, we declare the header of the *send*() and *receive*() JavaScript function that are used to communicate the GUI with the server but leave its body empty to encode them later, outside EJsS. Afterwards, we call the still uncoded *send*() function in the functions that have to send the already built messages to the server. And finally, after disabling the EJsS export option that obfuscates the JavaScript code generated by EJsS, we wrap the EJsS simulation (in our case used to develop the laboratory GUI).

After the previous steps, we have an intermediate version of our GUI, named **_javascript_simulation.xhtml*, that needs to be extracted from the *zip* file generated by EJsS and edited to 1) let it use the socket.io module in the GUI-sever communication, 2) connect the GUI with the WebSocket already opened by the server, 3) set up the event that will let the GUI recieve the messages from the server, and 4) implement the body of the previously uncoded *send*() and *receive*() routines. In this regard, *send*() uses the manually encoded WebSocket to emit the information that has to be sent and *recieve*() interprets/unpacks the messages that will be received after having setup its corresponding event.

Finally, and in order to be able to succesfully use the recently modified HTML5 + JavaScript file as the webpage of our laboratory, and load it within the server application, we have to modify it again to make the variable *_isEpub=false*.

3. LABORATORY EXAMPLE: A PID CONTROLLER FOR A VERTICAL MONO-ROTOR

Following the described strategy, we have developed a remote laboratory for controlling, using a Proportional/Integral/Differential (PID) controller, the height achieved by a vertical-moving mono-rotor. This section describes[16] the main characteristics of the plant under study and of the specific software elements (GUI and controller) developed for it.

[16] Additional information will be available at the ISCAR Lab webpage: http://www.dacya.ucm.es/isalab/index.html

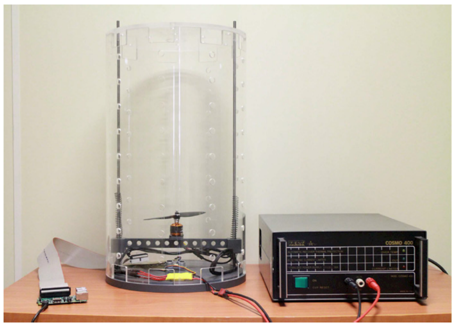


Fig. 7. Elements of the vertical mono-rotor

3.1 *The Vertical Mono-Rotor*

Our experimental plant, shown in Fig. 7 and inspired by the work in Sanchez-Benitez et al. (2012), is a moto-rotor that consists of a brushless motor that is coupled to a propeller and powered by a 36A Electronic Speed Controller (ESC), regulable through a Pulse-Width Modulated (PWM) signal. The ESC is powered by the DC Cosmo 400 by Kert power supply, which is able to provide a voltage/intensity of up to 13V/30A. The moto-rotor is attached to a horizontal-plate that is held by two vertical slides that constraint the movement of the plate within the vertical direction. The propeller thrust, and therefore the horizontal frame height, is regulated by the turning speed of the motor. Therefore the rotor will place itself at a different height according to the PWM signal that is applied to the ESC.

The horizontal-plate height is determined by the ultrasonic ranging sensor HC-SR043[17], which can be used to determine the distance to an object placed between 2 and 400cm in front of the sensor, with an accuracy up to 3mm. The ultrasonic signal sent by the sensor has to be triggered during 10μs through one of its input pins and the distance calculated from the high level duration of its output pin. Therefore the accuracy of the measurement is associated with the accuracy of the method used to determine the high level duration of its output signal.

Finally, its worth noting that, for safety reasons, all the lab elements except the power supply and Raspberry Pi are placed in a transparent protection cabin, which has been perforated to allow a correct air flow.

3.2 *The Controller Application*

The PID that the students apply to the vertical mono-rotor is implemented within a C controller application, following the structure presented in Fig. 4 and including a kernel module to determine the high level duration of the HC-SR043 output signal. In this particular case, the control signal applied to the ESC through a GPIO pin of the Rapsberry Pi is a PWM voltage output, which is generated using the *pwmWrite*() function of WiringPI.

The kernel module[18] is in charge of triggering the sensor and measuring, by means of hardware interruptions, the time that exists between the rising and falling edges of the HC-SR043 output signal, which is connected to a GPIO pin of the Rapsberry Pi. Just after each measurement, it writes the measured time in the kernel module CDF, which can be read, when required, by the controller application.

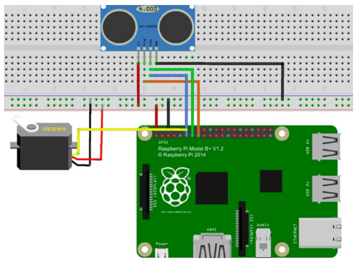
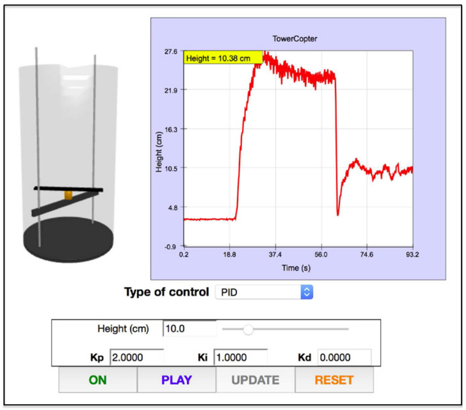


Fig. 8. Connections among the laboratory Raspberry Pi, sensor and ESC



Type of control PID

Height (cm) 10.0

Kp 2.0000 Ki 1.0000 Kd 0.0000

Fig. 9. Laboratory front-end

Finally, and to overview the connection scheme between the Raspberry Pi, ESC and HC-SR043, the reader is referred to Fig. 8.

3.3 *The GUI*

The GUI of the laboratory is presented in Fig. 9, where the evolution of the system is displayed in the top side of the GUI and the elements to interact with the controller application are presented underneath. When pushing the buttons at the bottom (from left to right): the GUI establishes the connection with the server, sends the starting/stop messages, sends the message with the values of the type of controller (Manual/PID) and PID parameters (proportional/integral/derivative constants Kp/Ki/Kd) that are selected in the GUI, or resets the GUI. The GUI automatically receives evolution signal messages and represents their values in the top figures.

4. CONCLUSIONS

This paper presents a new strategy to develop remote experiments for the control subjects of Systems Engineering and Automatica, based on the use of EJsS, the Raspberry

[17] HC-SR043 Datasheet: http://www.micropik.com/PDF/HCSR04.pdf

[18] We have followed this strategy, because the WiringPI polling software-interruption mechanism, which will require the controller application to determine when the interruption has occurred by checking a status interruption file, does not provide enough accuracy to determine the correct position of the vertical frame. To support our choice, we measure the standard deviation in the measure of the distance of an object placed at a fixed position with both methods. In ours, based on the use of a kernel module, the value of the standard deviation was 0.045cm, while in the other, based on the WiringPI polling software-interruption mechanism, it was 1.1cm.

Pi and Node.js. The first element was used to develop a HTML5+JavaScript laboratory front-end, accesible from mobile devices and computers. The second permitted us to run, in the same low-cost single-board computer, the laboratory server and controller. Finally, the third provided us with a JavaScript laboratory web sever and with the functionality to connect the lab front-end and server.

The combined use of all the developed elements lets us 1) close a real-time control-loop over the selected plant, 2) develop a GUI accesible from the smartphones and tables that the students carry everywhere, and 3) reduce the communication delays between the different elements by placing the server and controller in the same computer and using a lightweight efficient server for the lab webpage.

We are planning to improve our Node.js-based server to let it 1) display the actual image of the plant, 2) include restricted and exclusive/collaborative access management, and 3) store the evolution of the variables of interest for analyzing them later. Besides, we want to explore the possibility of 1) completely define the GUI within EJsS and of 2) using Node.js to connect the GUI with other types of controllers, to take advantage of its multi-platform possibilities. Finally, we are thinking of introducing this new laboratory in some Control subjects of the Universidad Complutense of Madrid during this academic year.

ACKNOWLEDGEMENTS

The authors want to thank D. Sanchez Foces for his support in setting up the experiments in the labs.

REFERENCES

Alhalabi, B., Marcovitz, D., Hamza, K., and Hsu, S. (2000). Remote labs: an innovative leap in the world of distance education. In *Proceedings of the 5th IFAC Symposium on Advances in Control Education.*

Besada-Portas, E., Lopez-Orozco, J.A., Aranda, J., and de la Cruz, J.M. (2013a). Virtual and remote practices for leaning control topics with a 3DOF quadrotor. In *10th IFAC Symposium Advances in Control Education (ACE).* Sheffield, UK.

Besada-Portas, E., Lopez-Orozco, J., de la Torre, L., and de la Cruz, J. (2013b). Remote control laboratory using EJS applets and TwinCAT programmable logic controllers. *IEEE Transactions on Education*, 56(2), 156–164.

Besada-Portas, E., Lopez-Orozco, J.A., de la Torre, L., and de la Cruz, J.M. (2012). EasyJava Simulations meets TwinCAT: Remote real-time control experiments using programmable logic controllers. In *9th IFAC Symposium on Advances in Control Education.*

Chacon, J., Vargas, H., Farias, G., Sanchez, J., and Dormido, S. (2015). EJS, JiL and LabVIEW: How to build a remote lab in the blink of an eye. *To appear in IEEE Transactions on Learning Technologies.*

Chang, G., Yeh, Z., Chang, H., and Pan, S. (2005). Teaching photonics laboratory using remote-control web technologies. *IEEE Transactions on Education*, 48(4), 642 – 651.

Chaos, D., Chacon, J., Lopez-Orozco, J.A., and Dormido, S. (2013). Virtual and remote robotic laboratory using EJS, Matlab and LabVIEW. *Sensors*, 13, 2595–2612.

de la Torre, L., Guinaldo, M., Heradio, R., and Dormido, S. (2015). The ball and beam system: a case study of virtual and remote lab enhancement with Moodle. *To appear in IEEE Transactions on Industrial Informatics.*

de la Torre, L., Heradio, R., Jara, C.A., Sanchez, J., Dormido, S., Torres, F., and Candelas, F.A. (2013). Providing collaborative support to virtual and remote laboratories. *IEEE Transaction of Learning Technologies*, 6(4).

de la Torre, L., Sanchez, J., Sanchez, J.P., Yuste, M., and Carreras, C. (2011). Two web-based laboratories of the fisl@bs network: Hookes and snells laws. *European Journal of Physics*, 32, 571–584.

Dormido, R., Vargas, H., Duro, N., Sanchez, J., , Dormido-Canto, S., Farias, G., Esquembre, F., and Dormido, R. (2008). Development of a web-based control laboratory for automation technicians: The three-tank system. *IEEE Transactions on Education*, 51, 35–44.

Esquembre, F. (2004). Easy java simulations: A software tool to create scientific simulations in java. *Computer Physics Communications*, 156(6), 199–204.

Fabregas, E., Farias, G., Dormido-Canto, S., Dormido, S., and Esquembre, F. (2011). Developing a remote laboratory for engineering education. *Computers & Education*, 57, 1687–1697.

Farias, G., Cervin, A., Arzén, K., Dormido, S., and Esquembre, F. (2008). Multitasking real-time control systems in easy java simulations. In *17th IFAC World Congress.*

Farias, G., Keyser, R.D., Dormido, S., and Esquembre, F. (2011). Developing networked control labs: A matlab and easy java simulations approach. *IEEE Transactions on Industrial Education*, 57, 3266–3275.

Grau, A. and Bolea, Y. (2008). Remote laboratory for control engineering degree. In *Proceedings of the 17th World Congress.* The International Federation of Automatic Control, Seoul, Korea.

Lazar, C. and Carari, S. (2008). A remote-control engineering laboratory. *IEEE Transactions on Industrial Electronics*, 55, 2368–2375.

Salzmann, C. and Gillet, D. (2007). Challenges in remote laboratory sustainability. In *International Conference on Engineering Education.* Coimbra, Portugal.

Sanchez-Benitez, D., Besada-Portas, E., de la Cruz, J.M., and Pajares, G. (2012). Vertical rotor for the implementation of control laws. In *9th IFAC Symposium on Advances in Control Education.*

Sivakumar, S., Robertson, W., Artimy, M., and Aslam, N. (2005). A web-based remote interactive laboratory for internetworking education. *IEEE Transactions on Education*, 48(4), 586–598.

Stefanovic, M., Cvijetkovic, V., Matijevic, M., and Simic, V. (2011). A LabVIEW-based remote laboratory experiments for control engineering education. *Computer Applications in Engineering Education*, 19, 538–549.

Vargas, H., Sanchez, J., Duro, N., Dormido, R., Dormido-Canto, S., Farias, G., and Dormido, S. (2008). A systematic two-layer approach to develop web-based experimentation environments for control engineering education. *Intelligent Automation and Soft Computing*, 14, 505–524.