Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Scalable Remote Experiment Manager

Matej Rábek and **Katarína Žáková**

Faculty of Electrical Engineering and Information Technology
Slovak University of Technology in Bratislava
Ilkovičova 3, 812 19, Bratislava, Slovakia
(e-mail: matej.rabek@stuba.sk)

Abstract: This paper describes an online laboratory system prioritizing modular and generalised solutions. The system provides a unified user interface to access, manage and control remote experiments. At the same time it does not put up restrictions limiting usable control algorithms or even simulation environments. Several connected devices are integrated with the use of Matlab and Scilab software so users can create their control block diagrams in either Simulink or Xcos. It is even possible to declare variables within these diagrams and then initialize them through the system's user interface. A set of commands was selected to represent the possible operations a device can implement. This serves to optimize the communication, since each of these commands is interpreted by a specialized python or shell script transferring all the user inputs and other necessary data either directly to the device or to a running instance of a simulation environment. System's functionality is demonstrated on a remote experiment based on levitating ball in a vertical tube. This method of control makes the system highly scalable as the new experiments can be added while the system is deployed.

Keywords: Virtual and remote labs; Internet based teaching of control engineering; Control education using laboratory equipment.

1. INTRODUCTION

Online control laboratories make the educational process much easier. With the advances of recent years they are becoming more and more versatile in the field of control algorithms. Enabling the use of various simulation environments lets the users experiment and learn new software solutions.

It is also imperative that the system is easily scalable. Adding new remote experiments or adjusting functionality is an integral part of a well-functioning online laboratory. The connected devices should not be limited to a single type of experiment. Utilizing their whole potential maximises their educational effectiveness. Finally, a unified interface to access multiple different remote experiment must be provided to assure a seamless user experience.

Following the results and applications in the field of the remote laboratories one can notice that there were already initiatives to build a remote lab management system. See e.g. (Stiubiener et al., 2006), (Harward et al., 2008), (Hardison et al., 2008), (Orduña et al., 2015), (Henke et al., 2016), (Uhlmann et al., 2018) and (Galan et al., 2019). They have various functionalities and various levels of generality. Our aim was to build a system enabling scalability and modularity in the used hardware, simulation environments and control algorithms.

An emerging trend in remote experiment development is the utilization of small singleboard computers such as Raspberry Pi or Beaglebone Black. They have been implemented in the online laboratory UNILabs (Sáenz et al., 2015). This approach, while energy and cost efficient, did not meet the necessary hardware requirements to run software with such high demand for resources as MATLAB at sufficient speed. At this moment, it only works with single-purpose experiments, which would defeat the objective of generalisation for this architecture.

2. MODULAR LABORATORY

Scalability of a system goes hand in hand with its modularity. Augmentation and expansion of an online laboratory often requires downtime. If the system is assembled from discrete modules on both the hardware and software levels, this issue can be resolved.

2.1 Software modularity

This paper describes design choices made in a specific online laboratory implementation described in (Rábek and Žáková, 2017). On the software level, the online laboratory system consists of several modules, each responsible for a specific functionality such as reservation system or management of uploaded control algorithms. The overall web application is built with Laravel framework. Pingpong/modules [1] is a package developed for this framework to help maintain larger and more complex applications. Each module contains its own separate model-view-

⋆ This work has been supported by the Slovak Grant Agency, grant VEGA No. 1/0733/16 and the grant APVV SK-IL-RD-18-0008. This support is gratefully acknowledged.

[1] Currently this package is no longer maintained and was republished and re-organised as nwidart/laravel-modules, which now also supports tests.

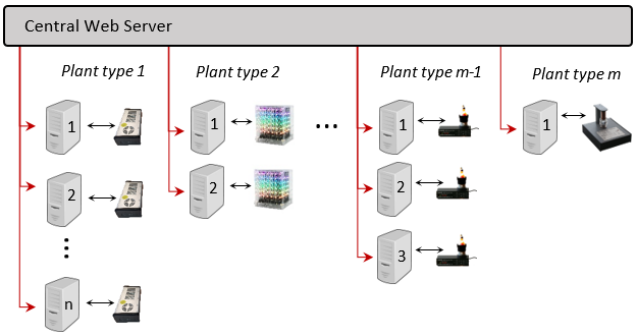

Fig. 1. System architecture depicting multiple devices connected to dedicated experiment servers.

controller implementation and can be enabled or disabled without influencing the overall functionality.

2.2 Hardware modularity

Hardware modularity stems from the chosen system architecture. A single server with a web application written in PHP running on it acts as a central node for the whole system. Users only directly access this server, which then relays their requests to dedicated servers for each of the connected devices to this online laboratory system (Fig. 1). It is still actually possible to access one of these dedicated servers directly, but that is only for development and debugging purposes and not intended as standard mode of operation.

The biggest challenge of achieving true hardware modularity is creating a unified process of integrating new experiments into the system. Since each experiment consists of a specific device and a dedicated server, an interface that facilitates communication between them has to be established. The system needs to treat these devices as black boxes in order to simplify their interactions. This proves to be rather difficult, since the connected devices differ from one another in many ways, such as number of inputs and outputs or even communication protocols. The only solution that preserves the freedom to integrate diverse range of experiments and at the same time access them from a unified interface is to implement a custom middleware which resolves these differences and bridges the system together.

3. UNIFIED INTERFACE

The server connected directly to a device acts as an interface between it and the central web server. The central server treats all the remote experiments equally and it is the experiment server's job to translate the received requests to commands for the device. Reading and formatting the output data is also handled by this server.

3.1 Commands

To control the connected device and its processes, four general commands have been chosen to represent all necessary actions that might need to be taken during the runtime of an experiment. These commands are:

- Init.
- Start.
- Stop.
- Change.

These commands are represented as executable scripts. Previously they were shell scripts but on newly connected devices, the Python programming language become the norm.

Each combination of a specific simulation environment and a device is handled by a complete set of these commands (with the exception of the optional init command). The PHP application running on the experiment server is capable of generating template files, which must then be altered specifically for the desired pair. Paths to these scripts are known by the web application. Depending on the type of experiment issued by the user, a correct script is executed and thus the experimentation process can begin.

Init The init command is the only optional command. This means, that it does not need to be necessarily implemented to manage the device's initialisation process, which might not exist. On the other hand, some devices require some time to initialize before they are able to perform the experiment. Generally, no arguments are necessary to issue this command.

Start This command starts the experiment. The input parameters it requires depend on the device, selected simulation environment and even on the control algorithm. The first task of the script is to either open the simulation environment or find a running instance and connect to it. After a connection is established, all the parameters are declared and initialized. Finally the experiment is started.

If the experiment is running in an open-loop mode, there is no need to launch it through the simulation environment. In this case, this script also takes over some more functionality as communication with the device and data collection.

Stop As the name suggests, this command terminates the experiment if for whatever reason it is decided, that the experiment should no longer be running. No input parameters are needed for this command.

Change The change command is quite similar to the start command. It changes the values of the variables determining the experimentation process during its runtime. Usually the input parameters are the same ones as for the start command.

4. COMMUNICATION AND DATA ACQUISITION

The complex online laboratory system is composed of multiple different layers that all communicate with each other (Fig. 2). Every one of them functions a little differently. Therefore, supplying the data in correct format is crucial.

4.1 Simulation Environments

Various simulation environments were successfully implemented to run on a single experiment server. Their main function is to provide a workspace to design and develop

control algorithms. These are then uploaded to the central web server. When an experiment is about to be executed, the dedicated server accesses the central storage and downloads a file containing the control algorithm. The appropriate start script loads the contents of the file to the environment and the experiment begins.

The downloaded file needs an appropriate extension in order to be read correctly by the corresponding software. This extension is not yet known by the experiment server at this point in the execution cycle since it treats all experiment requests the same way. The filename is generated randomly and then later a proper extension is appended by the executable script. As soon as the web server receives a notice that the experiment is finished, this temporary file is deleted so it does not unnecessarily fill up the hard drive.

Since the version R2015b Matlab software provides a way of creating a so called shared engine, which actually is a shared session that can be utilized by other processes (The MathWorks, 2015). This greatly improves the system's performance since it reduces the time needed for the software's initiation. This means, that a single instance of Matlab is at all times running in the background and handles all experiment requests. A python library, that facilitates the connection exists and was implemented to the system.

How these software solutions communicate with each other can be seen in Fig. 3.

Still, there are several restrictions using this method. The owner of the Matlab shared engine process must be the same user as the owner of the process trying to connect to it. Other difficulty is that some methods such as "set_param" can only be executed from within a software instance running in a desktop mode and not simply by a console application (The MathWorks, 2006).

To interface the device to the software two approaches can be explored. The first is to create an s-function, which as a single block represents the connected device. Other, more

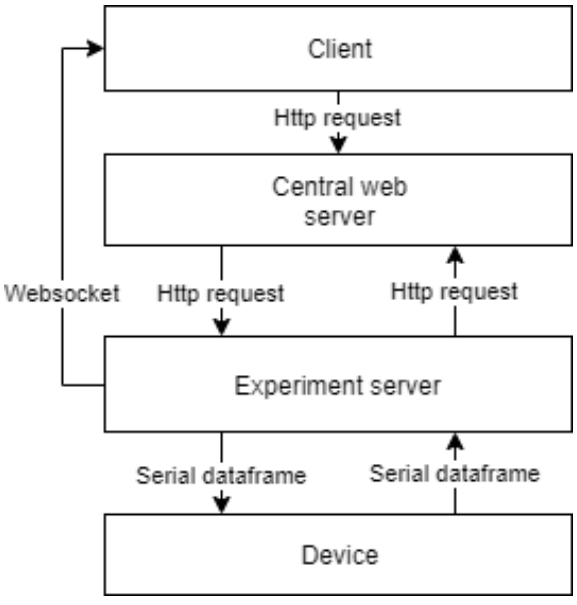

Fig. 2. Communication among system layers.

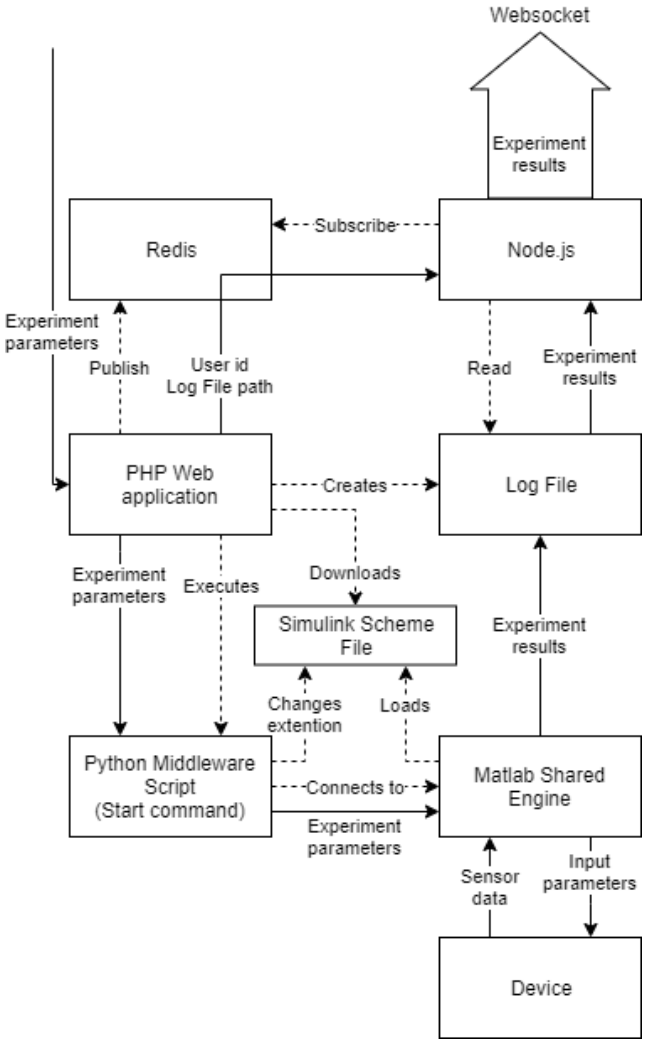

Fig. 3. Diagram of the functions and interfaces of a dedicated experiment server connected to a device.

generalised approach, is to use default blocks representing the serial interface, assuming the device communicates through USB or serial ports. While more universal, this method is less flexible to more unorthodox or legacy communication standards.

The Simulink "serial receive" block is designed to parse and interpret data obtained via the serial port. It is also possible to define the expected data structure of communication frames. As the frame header must be a character sequence that can not be found within the frame's payload, this greatly limits the variability of data that is sent by the device. Furthermore, even though the block is intended for binary data acquisition, the header can only be composed of printable ASCII characters, meaning that only values ranging from 32 (space) to 126 (˜) can be used. The restrictions placed on the frame terminator are greater still. There are only five specific options: Line feed ('\n'), carriage return ('\r'), combination of the two ('\r\n'), null ('\0') or no terminator at all. On the other hand, this block offers a lot more functionality out of the box, such as sampling time definition, simulation blocking or handling of unavailable data state (The MathWorks, 2008). To send

data back to the device a separate "serial send" block is used with similar properties.

4.2 Data collection

Each device is unique and can use a completely different protocol to communicate. Some may provide the output data on a request-response or polling bases while others can send them according to the predetermined sampling frequency. Other than that, the simulation environments use different forms of data storage. Characteristics such as these may pose a problem for an online laboratory system. To ensure compatibility, a persistent file is created by the web application as soon as the experiment request is received. It is identified by the users id followed by a string of numbers generated from the current time and date to prevent duplicate file names. This filename is passed along the experiment parameters to the executable script and a Node.js server running in the background.

Depending on the issued experiment, the script can forwards this filename to the simulation environment along other experiment parameters. In case the experiment runs in open-loop mode and does not require feedback, the data acquisition is among other things handled by this custom script.

As soon as the experiment begins, the Node.js server is informed via Redis in-memory data structure store. When notified using the implemented publish/subscribe messaging paradigm, the Node.js server establishes a websocket link straight to the connected user. Subsequently, the file is periodically read in a predetermined time interval and its contents are transmitted via created websocket identified by user's unique id.

The contents of this log file are dictated by the output data values provided by the device. The names of the physical units which are measured during the experiment are also defined within a separate configuration file.

4.3 Accessing the Data

The output data stream via the websocket is established right as the first values are read from the device and written to the log file. A JavaScript library HighCharts generate an interactive chart that is adjusted and redrawn with every received datapoint.

Alternatively, after the experiment is finished, the contents of the log file are sent as a HTTP request to the central web server, which to this point did not receive any data other than experiment status confirmations.

This whole process is depicted in Fig. 4 in great detail.

A report is generated from the gathered measurements and saved to the database. It can be accessed, viewed and even downloaded in a .csv format.

5. EXPERIMENT PARAMETERS

The experiment is defined by its parameters. Users require the ability to affect the behaviour of connected devices. The definable characteristics range from experiment duration to custom variables declared within the scope of uploaded block diagrams. All of them should be easily accessible and appear the same even though they are stored on different levels within the application and influence various aspects of the overall system and its parts.

5.1 Parameter Categories

Three separate categories of parameters were defined according to their function and area of effect.

General parameters describe behaviour of every experiment and thus must always be defined. These are the experiment duration or its sampling rate. Identification parameters such as IP address of the dedicated experiment server or selected simulation environment also fall into this category. They are therefore hard coded within the application.

Device parameters are identical for every experiment that can run on a particular device. These might represent systems disturbance variables. They are stored in configuration files on the experiment servers. Their template is automatically generated once a new experiment server application is installed and a device is added using the provided user interface. These files are later modified by the experiment developer in order to complete the device integration. The stored data contain the parameter name, display label presented to the user, data type and optional default or placeholder value. There is also a possibility to predetermine only a set of possible values if there is only a limited number of options to choose from.

Scheme parameters represent all variables declared in the uploaded block diagrams. A good example are the separate P, I and D values in a PID controller. Created parameter names are stored in the central web server's database and can be set by the owner of the particular simulation scheme. Similarly as the device parameters, these can be of different data types and have placeholder values assigned to them. They might also be presented in a form of a combo box with predefined options.

5.2 Control Panel

After a reservation for a specific device was made, the user can access its control panel. Here, users assign values to all of the parameters. The presented form is generated specifically for each experiment. The user selects a supported simulation environment and chooses one of the device instances if more than one were reserved for that exact time. After filling in all the values for the general and device parameters, a simulation scheme can be chosen. Depending on the selected device and simulation environment, a possible set of options is presented to the user. Upon selecting the preferred combination from the available options, the rest of the form is rendered according to the parameters tied to this specific scheme or block diagram. This way the user is presented with a unified visual design for all parameters, even though they determine different aspects of the experiment. After pressing the start button, a notification appears informing the user that the experiment was issued successfully if no problems during the execution occurred. A chart displaying the resulting data begins to render as soon as the device responds with

first measurements. With new data arriving periodically throughout the duration of the experiment, the chart is redrawn so it contains all available information.

5.3 Controller Management

The user interface to define the control algorithms was also designed and implemented in such a way, so that it appears identical regardless of selected device and even simulation environment. As it was previously stated, the central web server does not differentiate between implemented software solutions and even connected hardware devices.

Here users with administrative privileges create, modify and delete control algorithms. First a device type and a simulation environment pair is chosen. Then the user selects a controller file to be uploaded to the server and names the controller. There is also an option to provide a detailed description and even an image to better explain the block diagram. Finally, if necessary, the simulation scheme variables are introduced to the system.

5.4 Synchronisation

In order to alleviate the traffic between the central web server and the experiment servers, the stored device parameters are inserted to or updated in the central database. The central database also serves other purposes in addition to caching parameters for each experiment. It holds information on registered users, reserved devices and even experiment logs.

The synchronisation of data between serves can be performed by any authorized user with proper privileges. While the experiments are being synchronized, multiple tests and checks are performed. For instance, the databases are queried and Node.js servers tested. Also if the device was turned off or disconnected, it will not show up as available anymore after the synchronisation is finished.

6. CONNECTED DEVICE

The experiment chosen to demonstrate the functions of this online laboratory system is based on air levitation. A ping-pong ball is suspended in a transparent cylindrical tube. Its position is determined by a fan located in the base of the tube. A feedback is generated by a laser proximity sensor mounted on top of the fan. The behaviour of these electronic components is controlled by an Arduino Uno board, which communicates through USB port.

The experiment was inspired by similar implementations around the world (Jernigan et al., 2008), (Berisch et al., 2012), (Chaos et al., 2019), (Chołodowicz and Orłowski, 2017), (Uhlmann et al., 2018), (Dellah et al., 2000) and (Kuzhandairaj, 2018), but was also designed to mitigate few imperfections from which these suffer. The biggest difference by far from other devices created in this fashion is the use of a laser proximity sensor. At the time of building the device, the most commonly used sensors in comparable solutions where infrared and ultrasound. These come with their own set of flaws and inaccuracies stemming from the technology limitations. In any and all regards excluding its cost, the VL53L1X surpassed the competition (VL5, 2018).

This simple plant is controlled by one input variable - PWM signal generated by the Arduino board. It provides two output variables, which are the position of the ball (vertical distance from the sensor) and also the fan rotation speed measured in revolutions per minute obtained from the fan motor. More details about the plant construction and inner workings can be found in (Rábek and Žáková, 2019).

7. CONCLUSION

The proposed assignment was to increase the number of usable experiments in the online laboratory system while not sacrificing a unified user interface. Every connected device should be fully utilized and not restricted by a single

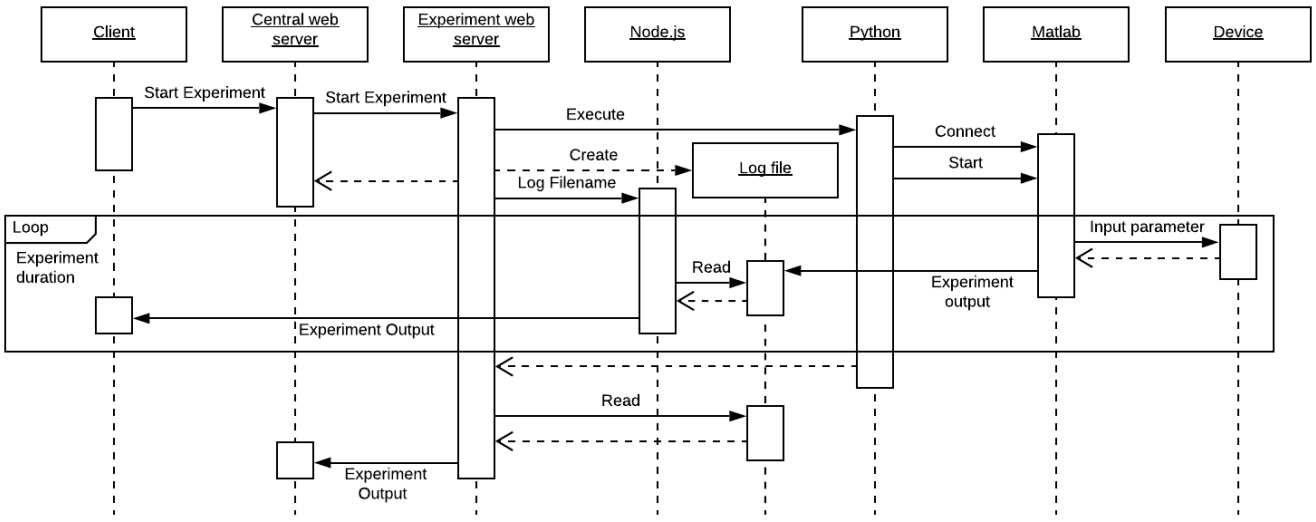

Fig. 4. Sequential UML diagram of an experiment lifecycle.

algorithm. This approach can potentially save resources and augment the educational potential of the system at the same time.

A high degree of modularity was achieved on both the software and hardware levels to prepare the system for potential future modifications and expansions. The implementation of a multitude of different technologies assures that every task is completed by a solution most suited for it. The various interfaces facilitating communication ensure overall compatibility and scalability. The process of implementing a new experiment is made easier by the generalised commands represented by executable scripts, which attach the often specific devices to robust simulation environments. Log files store the output data and along websockets are used to stream information to the user as the experiment is still running. A unified user interface to declare and initialize experiment parameters provide a comfortable way to set up and execute different kinds of experiments. The process of synchronisation detects malfunctions and secures the system's stability.

To test the integration process and overall functionality of the system a new device was developed from scratch. Even though this air levitation experiment was designed with this particular system in mind, its integration highlights the achieved modularity. It can be accessed from a unified interface and runs experiments in both open-loop mode using a simple python script and a closed-loop implementation with Matlab/Simulink.

ACKNOWLEDGEMENTS

Authors thank to Michal Galovič for his initial steps in the development of the introduced system.

REFERENCES

(2018). *A new generation, long distance ranging Time-of-Flight sensor based on ST's FlightSense^{TM} technology.* STMicroelectronics. Rev. 3.

Berisch, G., Donath, H., Heinrich, F., Huebener, I., Lipke, T., Mende, T., Schriefer, M., Schulte, H., and Zajac, M. (2012). Design and development of a low cost rapid control prototyping system applied to an air suspension system. *IFAC Proceedings Volumes*, 45(11), 194–199.

Chaos, D., Chacon, J., Aranda-Escolástico, E., and Dormido, S. (2019). Robust switched control of an air levitation system with minimum sensing. *ISA Transactions*.

Chołodowicz, E. and Orłowski, P. (2017). Low-cost air levitation laboratory stand using matlab/simulink and arduino. *Pomiary Automatyka Robotyka*, 21.

Dellah, A., Wild, P., and Surgenor, B. (2000). A laboratory on the microprocessor control of a floating ping pong ball. *age*, 5, 2.

Galan, D., Chaos, D., De La Torre, L., Aranda-Escolastico, E., and Heradio, R. (2019). Puzzlex: An online experimentation environment for control engineering labs. 69–73. doi:10.1109/EXPAT.2019.8876580.

Hardison, J.L., DeLong, K., Bailey, P.H., and Harward, V.J. (2008). Deploying interactive remote labs using the ilab shared architecture. In *2008 38th Annual Frontiers in Education Conference*, S2A–1–S2A–6.

Harward, V.J., del Alamo, J.A., Lerman, S.R., Bailey, P.H., Carpenter, J., DeLong, K., Felknor, C., Hardison, J., Harrison, B., Jabbour, I., Long, P.D., Mao, T., Naamani, L., Northridge, J., Schulz, M., Talavera, D., Varadharajan, C., Wang, S., Yehia, K., Zbib, R., and Zych, D. (2008). The ilab shared architecture: A web services infrastructure to build communities of internet accessible laboratories. *Proceedings of the IEEE*, 96(6), 931–950.

Henke, K., Vietzke, T., Hutschenreuter, R., and Wuttke, H. (2016). The remote lab cloud "goldi-labs.net". In *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 37–42.

Jernigan, S.R., Fahmy, Y., and Buckner, G.D. (2008). Implementing a remote laboratory experience into a joint engineering degree program: Aerodynamic levitation of a beach ball. *IEEE Transactions on Education*, 52(2), 205–213.

Kuzhandairaj, J.C. (2018). Development, control and testing of an air levitation system for educational purpose.

Orduña, P., Gómez-Goiri, A., Rodriguez-Gil, L., Diego, J., López-de-Ipiña, D., and Garcia-Zubia, J. (2015). wcloud: Automatic generation of weblab-deusto deployments in the cloud. In *Proceedings of 2015 12th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 223–229.

Rábek, M. and Žáková, K. (2017). Online laboratory manager for remote experiments in control. *IFAC-PapersOnLine*, 50(1), 13492–13497.

Rábek, M. and Žáková, K. (2019). Building of the fan driven ball levitation system. *IFAC-PapersOnLine*, 52(27), 283–287.

Sáenz, J., Chacón, J., De La Torre, L., Visioli, A., and Dormido, S. (2015). Open and low-cost virtual and remote labs on control engineering. *Ieee Access*, 3, 805–814.

Stiubiener, I., Ruggiero, W.V., Silveira, R.M., Korolkovas, I., Skopp, S., and Meiler, C. (2006). Netlab: A framework for remote network experiences. In *Proceedings. Frontiers in Education. 36th Annual Conference*, 19–24.

The MathWorks, I. (2006). URL `https://www.mathworks.com/help/simulink/slref/set_param.html`.

The MathWorks, I. (2008). Serial receive. URL `https://www.mathworks.com/help/instrument/serialreceive.html`.

The MathWorks, I. (2015). matlab.engine.shareengine. URL `https://www.mathworks.com/help/matlab/ref/matlab.engine.shareengine.html`.

Uhlmann, T.S., Sbais, P.H.D.S., and Mendes, L.A. (2018). Elsa-py-open-source extension for communication and data management in interactive remote experimentation under ilab shared architecture. In *2018 13th APCA International Conference on Automatic Control and Soft Computing (CONTROLO)*, 247–252. IEEE.