USING MATLAB AS A TOOL FOR FOCUSING THE LAB WORK IN ENGINEERING COURSES

Juan-Antonio Fernández-Madrigal

Systems Engineering and Automation Dpt., University of Málaga – Andalucía Tech (SPAIN)

Abstract

Computer programming is a fundamental requirement when applying many theoretical concepts of engineering in practice, even in non-computer-engineering areas. Unfortunately, the hardware devices used in the engineering work are rarely standardized in their programming frameworks; it is therefore very common that we need to provide background to the students of engineering courses on several programming languages and environments, which often consumes an important part of the time of a subject during the semester and is not directly related to the syllabus. Here we address this problem by concentrating all the programming required for this kind of subjects on Matlab®. We describe how we have established this software as a common and single prototyping workbench in some subjects for producing, quickly and easily, algorithms that manage a number of different hardware devices in the lab, and to analyse the data obtained from the lab experiments with the powerful tools already included in its scientific toolboxes, eliminating the need of changing from one environment to another at any phase of the exercise. This has optimized the learning curve and concentrated the limited lab time on the really important topics of our subjects, requiring a reasonable effort from the professor if he or she is an expert programmer.

Keywords: programming in engineering, Matlab, optimization of teaching.

1 INTRODUCTION

In modern graduate engineering courses, lab practices are a fundamental component in order for the students to grasp some difficult theoretical concepts, learn how to apply them in reality, and familiarise with the basic types of machines and environments that they will find in their future jobs. The majority of practical exercises included in engineering curricula at the university level involve the use of diverse and complex hardware devices in the laboratory (e.g., oscilloscopes, data acquisition equipment, microcontrollers, PLCs, robotic platforms, etc.) that, today, can be connected to a personal computer and configured and used through some programming environment.

Unfortunately, these programming environments come in as many flavours as manufacturers exist, which creates a problem when designing the timeline of the subject: the main concepts (both theoretical and practical) require, obviously, most of the time of the semester, but providing students with the basics for using each one of the devices may consume important periods of time as well. This issue is especially severe in highly multidisciplinary subjects, such as mechatronics [1], real-time systems [2], control systems engineering [3], or robotics [4], where the hardware devices are many and diverse, and there is neither a standardization of their programming environments nor an easy way of integrating their diversity in a given application. In this paper we are interested in how we can optimise the time devoted to providing the students of this kind of non-computer-engineering-centred subjects with the basics for using the lab equipment, in order to focus their learning (and the professor work) on the actually central theoretical and practical concepts.

Although there is no single programming environment today that can manage all the existing hardware diversity, two candidates must necessarily be considered due to their wide use in industry and academia, their support for hardware, and their increasing number of extensions to address problems from diverse disciplines: LabVIEW® [5] and Matlab® [6]. The LabVIEW® project began in 1983, being originally released only for Macintosh computers, and has remained as a proprietary product of National Instruments (NI) up to date. Its name is short for *Laboratory Virtual Instrument Engineering Workbench*, which is also its intended goal: to serve for designing and programming virtual instruments in the computer that connect to the available lab equipment, typically more rigid and limited in their user interfaces. Therefore it has a strong bias towards measurement and control applications. Currently it is commonly used for data acquisition, instrument control, and industrial automation. The main reasons for its success have been its graphical programming language, called G, suitable for non-programming experts, that is compiled into native code for execution in the

computer or in external controllers, and the wide variety of hardware devices that are supported. On the other hand, Matlab® was devised initially, around the 1970s, as a purely mathematical environment (its name comes from *Matrix Laboratory*) for the applied mathematics community in the university, but its potential for the market was realised soon, and a private company, Mathworks, continued its development proprietarily since the 1980s. Currently it has evolved into the tool of choice for intensive analysis and graphical representation of data coming from a wide diversity of sources, but it also includes extensions, called toolboxes, to connect to hardware devices that provide those data. It is shipped with powerful simulation packages, such as Simulink and Simscape. It is not as strongly connected to real-time control as LabVIEW®, though, because its support of hardware is not comparable and its programming language is interpreted and more complex for non-experts, but its toolboxes can be developed in more common languages, such as C [7], and it also offers the possibility to produce code to be executed in some external controllers.

We have chosen Matlab® for the problem at hand due to the following:

- A very relevant part of the lab work is to analyse and graphically represent data (collected from experiments). In this regard, Matlab® simply has no competitor in LabVIEW®.
- The same argument is valid for simulation: the capabilities of Matlab®/Simulink/Simscape (and other extensions) to run accurate and sound simulations of many physical processes and systems is state-of-the-art.
- LabVIEW® is an excellent choice for connecting to hardware equipment (especially if that equipment is also manufactured by NI); however, it is also easy for any expert programmer to extend Matlab® with new libraries, written in lower level languages, that connect to specific devices, as long as they are properly documented.
- The Matlab® real-time extensions only work for a few specific computer platforms, but the extension capabilities mentioned in the previous point also allow us to make reasonable real-time control in the same platform where Matlab® runs as long as it has a minimum computational power (like the modern PCs), since they can be programmed in such a low language as C. This is true, from a soft real-time perspective, even when using, instead of C, the interpreted language of the environment for that control, with the exception of systems with very quick dynamics (e.g., electrical circuits): many lab equipment, particularly the electromechanical one, can be controlled with acceptable polling times from Matlab®.
- We want to provide the students, many of them with no solid ground in computer programming, with an environment that does not create bad habits or inadequate mental structures in that respect. G is a dataflow visual programming language [8] that, although makes the development of algorithms less arid for non-expert programmers, does so through an approach that is rarely considered as efficient and well-devised as linear, textual languages used by expert programmers. Note that the main advantages of dataflow programming are to take benefit of parallelism and to provide a more direct perspective of the whole design, which are not so relevant concerns in most engineering lab scenarios. Although the programming language of Matlab® has not been adapted to modern paradigms, such as object-oriented [9], in the best ways, it is always possible to limit the use of controversial features. Its interpreted and weakly typed nature make the development and testing easy.
- Both Matlab® and LabVIEW® are proprietary software, which implies an important expenditure in the acquisition of lab licenses. (The latter typically involves the cost of acquisition of NI devices as well.) However, the former has a greater number of open-source and/or free competitors, for instance Octave [10], SciLab [11] or the programming language Python [12], increasingly used in academia. Many of these use the same programming syntax and even the basic API of Matlab®.

In the last years we have addressed the problem dealt with in this paper by concentrating the learning phase of all the programming skills required for our engineering subjects on Matlab®. We have summarised all the required background in an initial seminar, placed at the beginning of each subject, which has proven to be enough even for students with little programming background, and does not consume a long time from other parts. On the other hand, we have developed a number of libraries to connect to specific hardware devices, being the major drawback in that process the lack of documentation of those devices rather than the capabilities of Matlab® for being extended. These extensions have been mostly programmed in the Matlab® programming language or in C by the professor. With all this work we have optimized the student learning curve and concentrated the

limited lab time on the really important topics of the subjects. Furthermore, the work developed in one of these subjects serves to accelerate the progress of the students in subsequent ones.

The rest of the paper is structured as follows. In section 2 we describe the process by which we have set up Matlab® as our basic programming, modelling, simulation and data analysis tool in a number of graduate engineering subjects of the computer engineering, mechatronics and telecommunication engineering programmes at the University of Málaga [13]. In section 3 we describe the libraries that we have programmed as extensions of the original Matlab® system to connect it with diverse hardware devices: data acquisition cards, manipulator arms and mobile robots. Finally, in section 4 we summarise some conclusions and future work.

2 ADAPTING SUBJECTS TO THE APPROACH

During the last years we have implemented Matlab® as the main lab programming and working environment in several engineering subjects of the University of Málaga, listed in Table 1. They are representative of the complex teaching contexts mentioned in section 1, where the lab exercises are an important part of the semester and several, diverse devices must be used by the students. In this section we briefly describe these subjects and explain the main issues found during the process.

Table 1. Subjects where our approach has been implemented. The **Work load** column refers to three parameters that measure the amount of work involved in the subject: number of credits (either in the European Transfer Credit System or in the previously used Spanish system) / equivalent contact hours / equivalent total student work hours (only in ETCS).

Subject	Work load (see header)	Programme	School	Academic year	Typical no. of students
Robotics (ROB)	6 / 60h / 150h	Degree on Electronic Systems Engineering	Telecommunication Engineering	4th (last)	10
Data acquisition and control systems (DACS)	9 / 90h / -	Computer Science Engineer	Computer Engineering	3rd (last)	5
Computer control (CC)	6 / 60h / 150h	Degree on Computer Engineering	Computer Engineering	4th (last)	15
Mobile Robots (MOB)	5 / 37.5h / 125h	Master on Mechatronics Engineering	Industrial Engineering	1st	15

Some specifics of these subjects must be given for a proper understanding of our context. Data Acquisition and Control Systems (DACS) belonged to a programme that is in an extinction process these years due to the change of the previous Spanish university system to the new European Education Space (the last academic year when it was offered was Fall 2011-Spring 2012; it is still available for the students of the old degree but only for exams). A large part of its syllabus has been inherited by Computer Control (CC), including the adaptation to our approach. Also, a common characteristic of some of the subjects, namely DACS and Mobile Robots (MOB) is the heterogeneous academic origin of the students, who come from diverse programmes, rarely with sufficient programming background. In Robotics (ROB) and CC all students are homogeneous in origin, but only those enrolled in CC have enough programming skills, since it belongs to a computer engineering degree. Table 2 shows the mentioned aspects of the curricula, while Table 3 summarises the hardware and software tools used currently in the subjects, under our approach. Notice the intensive use of the Lego® Mindstorms NXT robots [14]; these robots have served as an invaluable, low-cost, multipurpose hardware tool that has contributed to the improvement of learning and motivation in the students, as we have reported in the last years ([15], [16], [17]).

Table 2. Heterogeneity of the students enrolled in the subjects of Table 1, and their background in programming.

Subject	Students come from…	Programming background of most students
ROB	All from the same degree.	Quite basic on Matlab® and on other programming languages.

DACS	Mainly from three different programmes in the same Computer Engineering School, but a few from the Telecommunication Engineering and Industrial Engineering Schools.	Quite strong on Matlab® and on other programming languages.
CC	All from the same degree.	Strong on common imperative languages (e.g., JAVA), basic on Matlab®.
MOB	Mainly from degrees in Industrial, Electrical, Mechanical and Telecommunications Engineering.	Quite basic on Matlab® and on other programming languages.

Table 3. Hardware and software currently used in the subjects of Table 1, after implementing our approach.

Subject	**Hardware**	**Software**
ROB	-Scorbot ER-V manipulator arms [18]. -Lego® Mindstorms NXT robots.	-Matlab®.
DACS	-ICPDAS PCI1202 data acquisition card [19]. -Simple RLC electric circuits. -Phywe DC motors by Lucas-Nuelle [21]. -Lego Mindstorms NXT robots.	-NXC [20]. -Matlab®/Simulink.
CC	-Lego Mindstorms NXT robots.	-Matlab®/Simulink.
MOB	-Lego Mindstorms NXT robots.	-Matlab®

Although our goal in the long term is to reduce all the software tools to Matlab®/Simulink/Simscape, in the case of the now extinguishing DACS, we could not fulfil that goal before it was substituted by the new academic programmes. Precisely DACS was the subject when we firstly faced the problem of the diversity of hardware and software tools in the lab: in the academic year 2002/2003 we tried to solve it through LabVIEW®, but the weirdness of the language for students coming from computer engineering degrees (with strongly acquired programming skills) and the necessity of using Matlab® for analysis, representation of data, simulation and design of control systems, made us to drop it.

In the case of ROB and MOB, the adaptation process towards the solely use of Matlab® deserves a more in-depth explanation. It began with the use of Matlab® as a tool for simulating some complex algorithms of mobile robots and representing the results graphically (namely, probabilistic localization and mapping, which require intensive use of matrix algebra [22]). Typically, mobile robots are programmed in high-level languages onboard, being C++ [23] the one of choice for such probabilistic problems, but the basic and heterogeneous background in that programming language of the students enrolled in both MOB and ROB, the complexities of the language when coping with such algorithms, the need for simulating the algorithms before implementing them in the actual robot, and also the need for an easy graphical representation of the data, made inadvisable to include it as a programming tool. For implementing algorithms onboard the Lego® robots, we chose NXC, a variant of C, much simpler than C++, and the BricxCC IDE [24]. In addition, in the case of ROB we also employed the ACL (Advanced Control Language), provided by the manufacturer of the Scorbot robots, for developing the manipulator arm lab exercises. The main problem with ACL programming is that it is obsolete since many years ago (as the model of the Scorbot robot that we have), and its programming environment is a simple serial console connected to the robot controller, where writing and debugging programs is really inefficient and frustrating; acquiring new manipulator arms involves more than one order of magnitude the budget of buying Lego® robots, thus we are stuck with this equipment.

In the last academic year (2013-2014) we decided to make an extra effort to reduce the number of software tools in both MOB and ROB, which led to the results presented in this paper. We implemented a Matlab® library on top of an existing one [25] that is able to communicate to the Lego® robot and control it remotely through an API very similar to the one provided by NXC, eliminating the

need for NXC and having all the computational power of Matlab® running in a PC compared to the interpreted NXC code running in the robot microcontroller. In the case of ROB, we also developed, through inverse engineering, a complete Matlab® library for controlling the Scorbot ER-V robots on-line from the PC, through the serial link, thus hiding the ACL language and providing the modern Matlab® IDE for code editing and debugging, benefiting from the power of a PC for computations (for instance, geometrical calculations related to the inverse and direct kinematic models of the robot).

In both MOB and ROB, this adaptation work has involved programming new libraries for Matlab®; we have found that for our mobile and manipulator robots the real-time requirements for performing a good control loop are satisfied in a modern PC (although, of course, it is not as efficient or predictable as implementing the algorithms onboard of the robot controllers).

In the case of CC (and, previously, DACS), particularly for the data acquisition exercises, we developed a new library for the connection with the data acquisition card. We did not use inverse engineering in this case, unlike for the very old Scorbot robots, that lacked all the documentation about the communication protocol; in the case of the acquisition card we actually had the API of the Windows® drivers, developed by the manufacturer of the card, thus all we did was to develop a Matlab® wrapper for that API. Since the card API is prepared for interfacing with C programs, we developed the wrapper in C, using the facilities of Matlab® for that.

In summary, the adaptation process to our approach has involved the development of new Matlab® libraries and then the adaptation of previous, not-entirely-Matlab®-based timetable of the subjects. Only one expert programmer (the professor) was required to carry out these tasks. An estimate of the time spent in this is shown in table 4. The benefits for the students, mainly coming from the increased time available for the relevant goals of the subjects and from the better fitting of the subject requirements to their backgrounds, are difficult to quantify, but qualitatively seem obvious.

Table 4. Estimate of the time spent by the professor (expert programmer in both C and Matlab®) to develop the libraries.

Library	**Languages and tools**	**Estimated development time**
Scorbot ER-V manipulator arms .	-Matlab® -RS232 serial monitor for inverse engineering	1 month
Lego® Mindstorms NXT robots.	-Matlab® -Lego® Matlab library	1 week
ICPDAS PCI1202 data acquisition card.	-Matlab® -C (MS VStudio) -Documentation of the API of the driver	1 week

3 CONNECTING MATLAB® TO HETEROGENEOUS LAB EQUIPMENT

We have developed a number of libraries of functions for Matlab® in order to connect to diverse equipment in our lab, namely: the ICPDAS PCI1202 data acquisition card, the Scorbot ER-V manipulator arms, and the Lego® Mindstorms NXT robots. In this section we describe with more detail each of these libraries. Creating new libraries for other devices should be straightforward just using the same techniques explained here, as long as the required documentation is available.

3.1 Interfacing Matlab® with the ICPDAS PCI1202LU data acquisition card

Fig. 1 shows the ICPDAS PCI1202LU data acquisition card, which connects internally to the PCI bus of the computer, and the external cable and screw board that connect it to the experimental scenario.

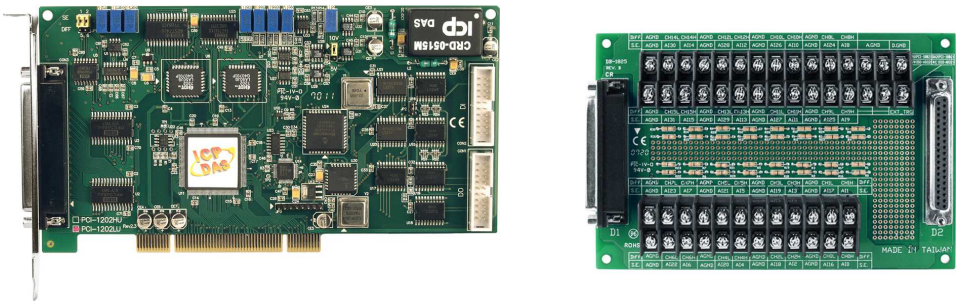

Fig. 1. Left) ICPDAS PCI1202LU data acquisition card (internally plugged-in to the PC). Right) External DB-1825 screw board connected to the internal PCI1202 through a cable, not shown.

This acquisition card has very interesting features for subjects involving data acquisition, real-time control of physical processes, and digital control (automation). We provide the students of DACS with a Matlab® set of functions that expose its capabilities for analog input/output exclusively. This is a summary of the features of this card that are relevant for our purposes:

- 32 analog inputs (single-ended) or 16 (differential) with [-10v,+10v] maximum input range, regulable in different ranges (e.g., [0v,2.5v], [-5v, +5v], etc.).
- 2 analog outputs in the [-10v,+10v] fixed range.
- 110,000 samples per second max. (i.e., we can sample dynamical systems with a minimum polling period of 9.1 microseconds).
- 12 bit resolution, from 0 to 4095, both in analog inputs (ADC) and outputs (DAC).

Our library is a set of functions written in C that include both the card driver and the Matlab® mex interfaces, in order to link the latter with the former. They are compiled with the compiler that Matlab® includes for *mex* (i.e., Matlab-external) files, producing native code that can be called from the Matlab® environment. For every function, we also provide a Matlab® code file that contains its documentation, in such a way that the `help` command in Matlab® displays it. The list of developed functions is in Table 5. Others can be easily added simply starting from their source code

Table 5. Functions developed for connecting Matlab® to the PCI1202 acquisition card.

Function name (prefixed with PCI1202_*)	**Summary**	**Uses**
InitDriverAndBoard, CloseDriverAndBoard	Initiates or closes the driver and the acquisition card.	Must be called before other functions, to initiate the card driver, and after the session is over, respectively.
ErrorText	Gets the description of an error code.	Transforms numeric error codes from the driver into textual messages understandable by the students.
SendDA, ReadAD	Sends output data through the digital/analog channels or reads input data from the analog/digital channels, respectively.	Can serve both to send a single voltage to an output analog channel and to send a number of voltages (a Matlab® vector of values) sequentially, at the maximum speed defined by the card and the execution speed of the code in the PC, to an output channel. The ReadAD function is analogous for reading from input channels.
SendAndRead	Simultaneously sends and reads data to/from analog channels (actually, interleaving a send with a read).	Given a vector of voltages to be output, send each one and sample a given input analog channel right after that send, collecting all the input values and timestamping them with the resolution of the CPU clock. This is useful for data sampling from dynamical physical systems subjected to an input signal defined by the user.

PID	Implements a PID controller.	Given the parameters of a PID controller [3], and the input and output channels for connecting to the system, performs a discrete PID control of it.

3.2 Interfacing Matlab® with the Scorbot ER-V manipulator arm

In this section we present the libraries developed entirely in Matlab® for the control of the Scorbot ER-V manipulator arms of our laboratory (see Fig. 2), manufactured by Intelitek. We have been able to write all the code in Matlab® through the use of the serial library included in it. The communication commands exchanged by the robot controller and the PC through the serial connection are mainly in ASCII text, and correspond to the ACL commands listed in the robot manual, but some inverse engineering of the serial link has been necessary in order to deduce communication parameters and handshake characters in between commands.

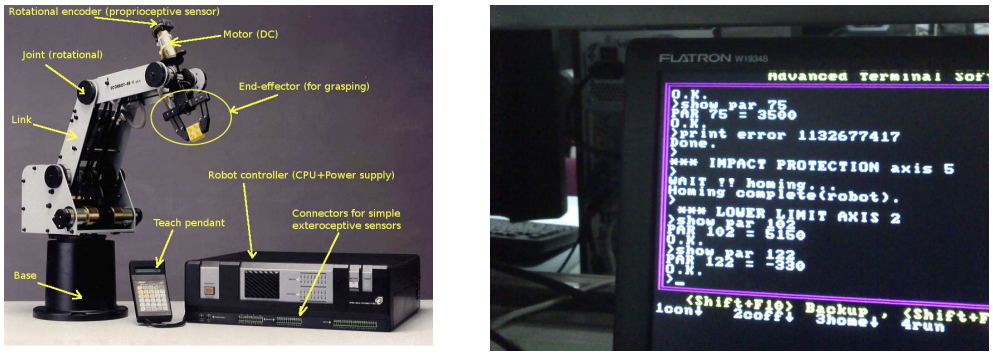

Fig. 2. Left) Scorbot ER-V plus robot (similar to the ER-V), with its teach pendant and the controller. Right) Primitive serial user interface shown in the screen of a PC connected to the robot controller through RS-232.

The main characteristics of this robot for our purposes are:

- 5 degrees of freedom (2 in the wrist) plus a grasping end-effector capable of lifting 1Kg.
- 0.5mm of position repeatability in the tip of the end-effector.
- Maximum velocity of 60cm/sec.
- Joint encoder sensors with 0.1 degree accuracy.
- Controller CPU: Motorola 68010 with 128K user RAM.

Table 6 lists the functions provided to the student for using this robot. Only a subset of the ACL language of the controller is exposed, mostly related to the storage and manipulation of both joint and cartesian *poses* (position+orientation) of the arm, and for motion commands. The controller provides better real-time response (actually, hard real-time) when programmed in ACL than if we use our library, which runs on a non-real-time operating system; however, the dynamics of the arm are slow enough for achieving soft real-time, and the benefits of not using the programming/debugging environment of the controller far outweigh that issue. In the case of our library, only one pose must be stored in the robot controller, being all the others created and managed in the PC; this is another advantage of our solution, since the native programming of the controller has strict limits on the number of poses (and variables, and programs) that can store.

Table 6. Functions developed for connecting Matlab® to Scorbot robots.

Function name (prefixed with Scb*)	**Summary**	**Uses**
Init, Finish	Opens or closes the RS-232 serial connection.	Must be called before programming the robot or after the session is over, respectively.

Home	Finds an initial position, using microswitches for detecting mechanical limits.	The robot must be homed at the beginning of each experiment, for calibration, and after lab work for leaving it in a safe position.
Gripper	Opens/closes the gripper.	Drop/grasp objects.
Speed, ChangeSpeed	Reads or sets the current speed of motion.	Perform tasks more accurately and safely.
CurrentPos	Gets the current robot pose.	From this pose, that can be defined externally through the teach pendant, the user can define others.
ChangePosJoint, ChangePosXYZ	Changes a pose in joint space or Cartesian space.	Define a new pose based in a previous one, changing some of the joint or Cartesian values, respectively.
Move	Move to a given pose.	Execute movements.
ClearMoveQueue	Empties the motion queue.	Sequential motion commands are queued until they are managed by the controller; this serves to clear the queue of pending commands.
Abort	Aborts pending motions.	Abort any pending or current motions, stopping the robot for the sake of safety.

3.3 Interfacing Matlab® with the Lego® Mindstorms NXT robot

The last library we have developed permits to command the Lego® Mindstorms NXT robot from Matlab® with a set of functions very similar to the ones used when programming the robot onboard with the NXC language. Fig. 3 shows the Lego® Mindstorms NXT robot. Its main features are:

- The main CPU is a 32bits ARM7 core @ 48MHz, with 128Kbytes for user programs and 64Kbytes for RAM variables. A second CPU of 8bits accesses some sensors and actuators.
- Bluetooth/USB communications.
- Up to 4 input connectors for sensors. Standard sensors: ultrasound (range), light, sound (microphone), and bumpers. In addition, the motor actuators have encoders that provide the angular position of the motor with 1 degree of resolution.
- Up to 3 output connectors for actuators, like DC motors that can be controlled in speed.
- Black&white display, 60x100 pixels, for both text and graphics.

The library is just a wrapper upon another one, called RWTH [25], both written entirely in Matlab®. This wrapper will be maintained unaltered even when we change to the new Lego® Mindstorms EV3 robot model. It provides a simple interface and all the numerical and computational power of Matlab® in the PC, while NXC is certainly limited in both aspects. On the other hand, the communication overhead incurred by the connection PC/robot through our library must be taken into account; it depends on the type of communication link, being USB preferred as long as the motion of the robot is not limited by the cable length. Table 7 lists the main functions developed in this library.

Fig. 3. The Lego® Mindstorms NXT robot used in our lab in two slightly different configurations, differing only in the placement of the ultrasound sensor, each one suitable for a different set of exercises.

Table 7. Functions developed for connecting Matlab® to Lego® robots.

Function name (prefixed with Lego*)	**Summary**	**Uses**
Connect, Disconnect	Creates or closes a connection with the robot.	Must be used before calling any other function and after the session is over, respectively.
SetSensorSound SetSensorLight, SetSensorTouch, SetSensorLowspeed	Indicates in which input port is connected each sensor.	Must be called for all the sensors mounted on the robot before executing any task.
Sensor, SensorUS	Reads sensor value.	Accesses to the sensor measurements.
OnFwd, OnRev, Off	Motion commands.	Move the robot forward or backward, or stop them.
ResetRotationCount, MotorRotationCount	Resets and reads motor encoder.	Consult the angular position of a motor encoder, or set it to 0.
ClearScreen, TextOut	Clears the screen and display a text.	Showing messages to the user.
ButtonPressed	Reads robot button status.	Interfacing with the user through robot buttons.

4 CONCLUSIONS AND FUTURE WORK

In this paper we have described the adaptation of different engineering subjects at the University of Málaga in order to minimise the time devoted to the programming background required to manage a diversity of hardware devices. This has allowed us to focus the teaching work (and the learning process of the students) on the central concepts, both theoretical and practical, of the subjects, and also to make them more amenable to students coming from different programmes.

The solution consists in concentrating all the software work on Matlab®. This implies to develop a number of libraries/extensions of the basic system that allow us the student to connect, configure and use the diverse hardware equipment we have in our labs. Since Matlab® includes powerful simulation tools and data analysis and representation functionality, it becomes the work center of the students in all the phases of their practical lab exercises, eliminating the need to learn any other programming language or tool. The development of the libraries can be carried out in a very reasonable time by an expert programmer. In addition to this programming effort, the professor must include at the beginning of the semester a seminar on Matlab® that includes the basics of this environment and programming tool. We currently have a four-hours (two-sessions) interactive seminar for that.

The work described in this paper has spread over more than a decade, including the development of libraries, the adaptation of the semester planning for the subjects, and the design and test of the practical lab exercises based on the new Matlab® framework. In the future we plan to extend the number of hardware devices covered, particularly to subjects on Real-Time Systems, which have special requirements that do not fit so well with Matlab®. In that case, we will rely on C language for implementing real-time critical parts of the algorithms.

REFERENCES

[1] Bolton, W. (2011). Mechatronics: Electronic Control Systems in Mechanical and Electrical Engineering. Pearson, 5th edition, ISBN 978-0273742869.

[2] Laplante, P.A., Ovaska S.J. (2011). Real-Time Systems Design and Analysis: Tools for the Practitioner. Wiley-Blackwell, 4th edition, ISBN 978-0470768648.

[3] Nise, N. (2011). Control Systems Engineering. John Wiley & Sons, 6th edition, ISBN 978-0470646120.

[4] Craig, J. (2013). Introduction to Robotics: Mechanics and Control. Pearson, 3rd edition, ISBN 978-1292040042.

[5] National Instruments (2014). LabVIEW product page. http://www.ni.com/labview/, retrieved 16 Sept. 2014.

[6] Mathworks (2014). Matlab product page. http://www.mathworks.co.uk/products/matlab/, retrieved 16 Sept. 2014.

[7] King, K.N. (2008). C Programming: A Modern Approach. W. W. Norton & Company, 2nd edition, ISBN 978-0393979503.

[8] Johnston W.M., Paul-Hanna J.R., Millar R.J. (2004). Advances in Dataflow Programming Languages, in ACM Computing Surveys, Vol. 36, No. 1, pp. 1–34.

[9] Wong, S. (2009). Principles of Object-Oriented Programming. Orange Grove Books, ISBN 978-1616100629.

[10] GNU (2014). Octave product page. http://www.gnu.org/software/octave/, retrieved 16 Sept. 2014.

[11] Scilab Enterprises (2014). Scilab product page. http://www.scilab.org/, retrieved 16 Sept. 2014.

[12] Dawson, M. (2010). Python for the absolute beginner. Course Technology PTR, 3rd edition, ISBN 978-1435455009.

[13] University of Málaga (2014). Academic programmes web page. http://www.uma.es/cms/base/ver/base/basecontent/4381/oferta-academica/, retrieved 16 Sept. 2014.

[14] Lego (2014). Lego Mindstorms product page. http://www.lego.com/en-us/mindstorms, retrieved 16 Sept. 2014.

[15] Cruz-Martín, A., Fernández-Madrigal, J.A. (2009). Using Lego Mindstorms NXT robots for control systems courses in undergraduate engineering programs, International Conference on Education, Research and Innovation (ICERI'09), Madrid (Spain).

[16] Cruz-Martín, A., Fernández-Madrigal, J.A., Galindo, C., González-Jiménez, J., Stockmans-Daou, C., Blanco, J.L. (2012). A Lego Mindstorms NXT approach for teaching at data acquisition, control systems engineering and real-time systems undergraduate courses. Computers & Education, vol. 59, no. 3.

[17] Cruz-Martín, A. (2014). Lego Mindstorms NXT: A useful and innovative approach for different educational levels. In Learning Environments. Technologies, Challenges and Impact Assessment, Nova Publishers, ISBN 978-1-62808-700-0.

[18] Intellitek (2014). Scorbot robots product page. http://www.intelitek.com/advanced-manufacturing/robotics/, retrieved 16 Sept. 2014.

[19] ICPDAS (2014). PCI1202 card product page. http://www.icpdas.com/root/product/solutions/pc_based_io_board/pci/pci-1202.html, retrieved 16 Sept. 2014.

[20] Hansen J.C. (2009). LEGO® Mindstorms NXT Power Programming: Robotics in C. Variant Press, 2nd edition, ISBN 978-0973864977.

[21] Lucas-Nuelle (2014). Lucas-Nuelle company page. http://www.lucas-nuelle.com, retrieved 16 Sept. 2014.

[22] Fernández-Madrigal J.A., Blanco J.L. (2012). Simultaneous Localization and Mapping for Mobile Robots: Introduction and Methods, IGI Global, ISBN 9781466621046.

[23] Stroustrup, B. (2013). The C++ Programming Language. Addison Wesley, 4th edition, ISBN 978-0321563842.

[24] BricxCC (2014). BricxCC home page. http://bricxcc.sourceforge.net/, retrieved 16 Sept. 2014.

[25] RWTH Aachen University (2014). RWTH Mindstorms NXT toolbox home page. www.mindstorms.rwth-aachen.de, consulted 16 Sept. 2014.