LAWRIS: A Rule-Based Arduino Programming System for Young Students

S. Arakliotis
Information & Comm. Systems Eng. Dept., University of the Aegean, Greece
icsd08005@icsd.aegean.gr

D. G. Nikolos
Educational Sciences & Early Childhood Education Dept., Univ. of Patras, Greece
dnikolos@upatras.gr

E. Kalligeros
Information & Comm. Systems Eng. Dept., University of the Aegean, Greece
kalliger@aegean.gr

***Abstract*—Since Arduino is the main entry-level platform to the world of electronic circuits and systems, there are many programming environments that try to ease the burden of Arduino textual programming on young primary students. Although block-based, most of these environments retain the imperative structures of textual programming languages, which are not easily comprehensible by such students. The proposed, in this paper, Learning-Arduino-With-Rules Introductory System (LAWRIS) tries to tackle this problem by adopting a rule-based approach: an Arduino-based pre-specified system can be programmed by constructing rules with very simple and intuitive jigsaw pieces, in a visual, web-based environment. Contrary to other programming environments for Arduino, the main logic of LAWRIS is implemented on the board side and only a configuration string is downloaded to it. Thus, LAWRIS features fast responses to both circuit input changes and program modifications, a very important characteristic for young students. It also imposes minimal overhead on the host system, allows web access of the programming environment, and has very low hardware cost.**

Keywords—rule-based programming; learning programming; Arduino; Blockly

I. INTRODUCTION

The Arduino platform has managed to become a standard in prototyping projects related to electronics and in making electronics accessible to beginners. It comes with the open source Arduino Software, a textual programming language based on Wiring and an IDE based on Processing [1]. Though Arduino enables beginners to transform their ideas into real projects, its textual programming language might not be suited for young learners. Learning to program can be a difficult task for beginners [2]. With commercial programming languages, beginners have to understand algorithmic solutions to problems, construct a mental model for the machine, learn the syntax of the language and interpret the names of the commands of the specific language. This is also the case with Arduino.

Scratch [3] and other programming languages based on blocks, gain interest in young student programming classes. Several attempts have been made to incorporate block based programming into physical computing, and specifically Arduino, including Splish [4], MiniBloq [5], Modkit [6] and Scratch4Arduino (S4A) [7]. Splish is a graphical environment for the Arduino platform that follows the flowchart/UML approach for implementing common programming structures. We think that flow control is not appropriate for young primary students. Minibloq provides a graphical alternative to the textual Arduino language, bringing it to the visual domain. Minibloq features a large selection of blocks but keeps the main logic of Arduino introductory programming with control structures and delays, an approach that may not be appropriate for young students. Also, the programmer must specify the corresponding port for each block, a fact that adds some difficulty to the environment. Modkit and Scratch4Arduino are integrating the Scratch programming structures into Arduino programming, having blocks from both worlds in their environment. However, the commands used for Arduino are similar to the original ones, e.g., to turn on an LED on port 10 the command reads “DigitalWrite PIN10 HIGH” (Modkit) or “digital 10 on” (S4A). We think that young students may not be able to create appropriate mental representations for such commands.

This paper presents Learning-Arduino-With-Rules Introductory System (LAWRIS), a visual web-based environment for teaching young primary school students programming with Arduino prototyping boards. The objective of the system is to introduce such students to physical computing. LAWRIS is not based on imperative programming structures but uses a conditional rule formalism that may be easier for very young students [8]. The proposed system consists of three parts: the user-interface that is based on the Blockly visual editor [9], a specific circuit that connects input and output devices to the Arduino, and the Arduino software, which implements the main logic of the system.

II. DESIGN CONSIDERATIONS

LAWRIS is a web-based learning platform, i.e., the user programs in the browser. As a web platform it can run in a browser window, in most computers that can communicate with Arduino. Beginner programmers need to be able to tinker with the system they program as much as possible. Immediate feedback is also very important [10]. LAWRIS is designed in a way that does not need the whole code to be downloaded to the Arduino as a separate step. Only a configuration string is sent to it. Without sending anything back to the host, the Arduino reconfigures its behavior and the results are immediately visible on the circuit. As a result, LAWRIS features really fast input sampling and immediate output reactions to inputs. Also, LAWRIS does not impose real-time processing on the host, constituting a viable solution for a variety of computer systems.

We used Blockly [9] as LAWRIS' drag and drop visual editor because: (1) it is an open source project, (2) it provides syntactically perfect code so that it would be easier for the young learners to tinker with their ideas without having to consider the syntactic details of a textual programming language, and (3) it has been successfully used for introducing young students to programming, mainly in the Hour of Code activities [11].

LAWRIS supports four input devices: a push button, a potentiometer, a sonar distance sensor and a photoresistor. It also

978-1-4673-9680-6/16/$31.00 ©2016 IEEE

supports four output devices: a common LED, an RGB LED, a motor and a piezo buzzer. These devices are selected not only because they are the most common I/O devices used in Arduino projects, but also because they are the building blocks of many projects made by young students [12].

LAWRIS hardware is designed to be straightforward and consists of the circuits that beginners normally use to connect each of the primitive devices to the Arduino. In that sense, it is easy to be constructed by the instructor. Ideally, the circuit could be constructed by the students with some guidance. This process would provide basic knowledge about electronics.

With LAWRIS, the Arduino is programmed with rules. Every rule is a triplet of input, output and optional operators (see below). The user can set an input to control any output, and should also specify the time for which the binding will be active. The beginner has only to combine input and output blocks. They cannot combine input blocks with input blocks or output blocks with output blocks. Thus, a robust jigsaw environment for programming is provided, where no syntax errors can occur. Also, an input can drive more than one output, but an output can be controlled by only one input. All the above create an elemental environment for introducing young students to Arduino.

The student can use any of the inputs to drive one output. Also, they can tweak the result using a set of operators like invert, threshold (the input affects the output only when surpassing a threshold, i.e. a percentage of its maximum value), or scale (the input values are scaled down by a user-specified rate, before driven to the outputs). All of the outputs can be further tweaked with a set of operators special to some output. The color of the RGB LED can be specified, the rotational direction of the motor can be inverted, the note that the Buzzer plays can be specified and the LED can be programmed to blink. These operators can be combined in a series, providing functionality for complicated projects. As with every feature of LAWRIS, the changes that the user makes are visible on the circuit with a click on a button.

All the block elements are color coded: input blocks are blue, output blocks are green, timing labels are red and all operators are olive green. Color coding helps student scaffold in learning the language. If they apply a binding correctly and see the result, they can apply a different binding using the same color scheme [13].

III. Implementation

LAWRIS implementation comprises the development of its three main components: (1) the hardware system, (2) the user-interface, and (3) the Arduino software. In the following, we provide the details concerning each of these components.

A. The Hardware System

As already mentioned, LAWRIS is an Arduino-based system. Specifically, it has been implemented using the Arduino Mega 2560 Rev3 board that was available in our Lab. Note though that, for decreasing the cost, simpler (and hence cheaper) boards, like the Arduino Uno, can replace the Mega, since both the available memory and the I/O pins suffice. However, a more advanced board like the Mega offers greater flexibility in the case that future system enhancements are needed.

In Fig. 1 the connections of the input devices to the Arduino board are shown. As can be seen, the wiring is pretty

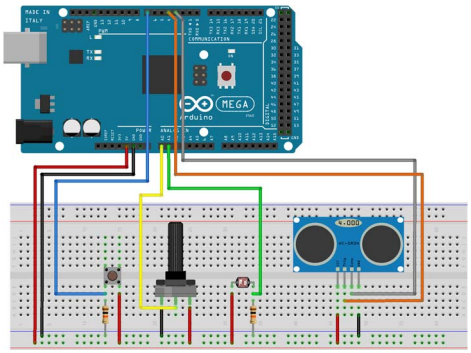

Fig. 1. Input device connections of LAWRIS.

standard; the push button and the sonar sensor are connected to digital I/O pins (configured as inputs), whereas the potentiometer and the photoresistor drive analog inputs. Please note that, if needed, simple digital I/O pins (without PWM output capability) can be utilized for the push button and the sonar, in order to free, for PWM output use, those currently exploited.

Fig. 2 shows the output device connections of our system. All four devices (LED, RGB LED, motor, piezo buzzer) are connected to PWM outputs, since some kind of analog control is required for each one of them. Specifically, the intensity of the two LEDs, the rotational speed of the motor, and the volume of the piezo buzzer are all controllable from the inputs. Again, for the two LEDs and the motor, standard connection circuitries are utilized (that of the motor includes an SN754410 H-Bridge and an external battery pack). The only "unusual" connection is that of the piezo buzzer, which uses two PWM output pins, instead of just one and the GND pin. The reason is that such a connection is required by the toneAC library, which is employed in our code and allows playing sounds of different frequencies with the piezo buzzer.

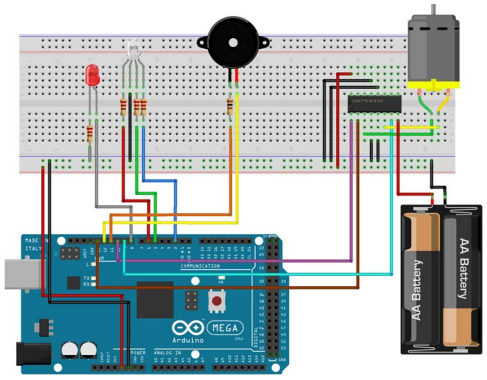

Fig. 2. Output device connections of LAWRIS.

B. The User-Interface

The user-interface of LAWRIS runs on a web browser and has been built using Blockly. The visual environment is simple and clean, as required by the age of its potential users, and includes a single "Play" button (see Fig. 4). A click on that button, after having described the desired system's behavior by connecting Blockly jigsaw pieces, causes the transfer of a character-based configuration string from the host computer to the Arduino board, which, in turns, results in the immediate change of the system's behavior. The structure of the implemented code is shown in Fig. 3.

Blockly code generation is pretty automated. The code for each block (jigsaw piece) was produced by using the "Block

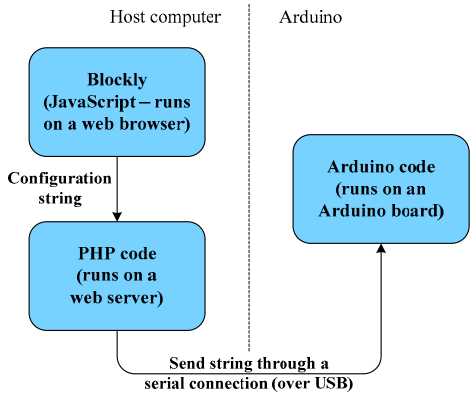

Fig. 3. LAWRIS code structure.

Factory" application. Then, all generated code was appended in a JavaScript file, which was subsequently included into a web page along with other predefined Blockly files. The JavaScript code for injecting Blockly into the web page is predetermined, as well. However, some extra code was needed for performing error checking; when the "Play" button is clicked, the user is notified about some fundamental errors that may exist in the drawn configuration. Those errors are: (1) no blocks have been used, (2) one or more blocks have been left unconnected, (3) a special function block has not been combined with the appropriate output device, and (4) a duration block is missing (see below and in Section IV).

The main task of the JavaScript code that runs under Blockly is to generate the configuration string for the Arduino board. Specifically, the function of each block is to augment the string with the appropriate character(s) (refer to the next paragraphs). For sending the string to the board, a serial connection between the host computer and the board should be established (of course, the physical connection is over USB). This can be easily done using a few commands of the host computer's operating system. For that reason, a web server was employed. Specifically, we used the WAMP stack, since our host computer was Windows-based. A small PHP code receives the configuration string that was created, and, after some small modifications (mainly, space removals), sends it serially to the Arduino board. This code is executed quietly (i.e., without any interaction with the user) on the web server.

The last thing that needs to be discussed is the structure of the configuration string. Each connection between an input and an output device (i.e., rule) is denoted by 5 characters, in the form: *<output (1 char)> <operator (3 chars)> <input (1 char)>*. A single character is used for every input or output, as follows: 'B' - push button, 'P' - potentiometer, 'S' - sonar sensor, 'L' - photoresistor, 'l' - LED, 'r' - RGB LED, 's' - piezo buzzer, and 'm' - motor. Note though that any other character could have been used for this purpose. The operator between two connected devices is defined by the first character of the corresponding field. Specifically, a simple connection, with no operator, is denoted by character 'A' (ASCII code in binary: 01000001). The presence of an operator is declared by setting one of the rightmost three zeros of this binary codeword (namely, the first one for invert, the second for threshold, and the third for scale). For example, if invert and threshold were chosen between an input and an output device, then the corresponding codeword would be 01000111 (= 71, i.e., character 'G'). Bit setting can be easily done by using the bitwise OR operation. The advantage of the described notation is that a single character suffices for representing any combination of the three available operators between an input and an output device. The last two characters of the operator field are used for storing the percentage value of threshold and scale (one character for threshold and one for scale). If neither threshold nor scale is chosen by the user, then the corresponding character is ignored by the Arduino code.

When a special function is used, some extra characters are added, after the five described in the previous paragraph. The first of those extra characters identifies the special function ('$' - LED blink, '^' - RGB LED's color, '*' - buzzer's sound pitch, and '%' - motor reverse). The characters that follow it (if any) correspond to the parameter of the special function. Specifically, a single character follows '$' and '*' indicating the speed of LED's blinking and the musical note that the buzzer will play, respectively, while six characters follow '^' denoting the color of the RGB LED in hexadecimal form. Motor's special function needs no extra characters.

Combining multiple rules together, as a set, is perfectly feasible in LAWRIS. Different rules are separated by character '!' in the configuration string. Multiple sets of rules can be specified as well, where each set may contain one or more rules. Every rule set should be accompanied by a duration block. Duration is denoted by two characters in the string, which correspond to the actual time, in seconds, that the rule set should be active. These two characters are followed by '-', which is the last of the characters describing a set. The end of the configuration string is indicated by character '.' .

It is important to note that the choice of not generating Arduino code but reconfiguring the system with a small character string, leads to a single, very brief communication between the user-interface and the hardware, per configuration. This feature distinguishes LAWRIS from other Arduino learning systems and, apart from the advantages mentioned in Section II, allows seamless system operation in case of setting and accessing the user-interface remotely, from a website.

C. *The Arduino Software*

The Arduino code is the heart of LAWRIS since it performs all of its functionality. In contrast with some other systems [7], the Arduino board in LAWRIS does not just read the inputs and drives the outputs but also performs all the necessary processing. Therefore, the two main requirements for the Arduino code are full functionality under any rule combination(s) and reconfigurability.

For fulfilling both the above requirements, it is necessary to dynamically identify the connections of every rule set. Rather than creating a special connection matrix, the configuration string is directly used for this purpose. After removing from the string all the additional information (special functions and duration) and keeping them in separate variables, the "bare" five-character rules are repeatedly checked in succession and the corresponding connection values are updated. The special functions are dealt with as additional output attributes (their application though is not always straightforward, as in the case of LED blinking, for which some extra coding was needed). The repeated cyclic evaluation (i.e., reading the input, applying the operators, updating the output) of a set's rules, lasts exactly as much as the configuration string specifies (to this end, a time-controlled while loop is used). After the specified duration, the next set of rules is processed in the same way.

Another important issue is that of the values that are read from the inputs and driven to the outputs. In order to allow every possible rule to be realized in LAWRIS hardware, a common range of values ([0, 255]) is used for all devices. All input values are mapped to this range, while all operators affect the same range, as well. As for the output devices, they get the required values with a final reverse mapping (from [0, 255] to the range specified by their manufacturer), if necessary.

Finally, apart from the toneAC library for the piezo buzzer, the NewPing library is used for controlling the sonar sensor. It is worth noting that all the above design choices confine the Arduino executable code to merely 13.4 Kbytes.

IV. Examples

The inexperienced user can create interesting projects easily. They can drag and drop commands (jigsaw pieces), grouped on the left side of the environment, to construct rules. In Fig. 4 the available input commands are shown. The depicted rule on the right, programs a motor that changes its speed according to the potentiometer for 60 seconds (other duration options are also available). Students can observe the outcome of their actions immediately after clicking on the play button. We believe that beginners will be motivated to program with the environment, since they can make real working projects easily.

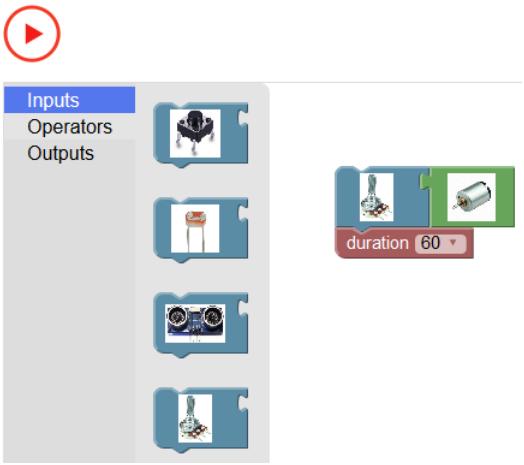

Fig. 4. LAWRIS programming environment and a rule that controls the motor using the potentiometer.

As they continue to use the proposed system, young students can create more complex programs. For instance, they can build a control system for an automatic door and its alarm. Suppose that they decide to use the push-button to activate the motor for 10 seconds, simulating in this way the door. After the 10 seconds of the first rule, they can program an LED to light brighter when something approaches the sonar distance sensor. If they want to use the RGB LED, LAWRIS provides them with the means to choose the color of the light (Fig. 5). They can also add a rule for the system to beep when something is too close. Beginners can split the program in different sets of rules. LAWRIS provides a framework for problem decomposition, a very important technique in programming.

Students are expected to work on the virtual environment and move over to the actual circuit to check its behavior. In the example of Fig. 5 we expect beginners to actually move their hands over the sonar and see what happens. In the following, they can change the threshold value to calibrate their system.

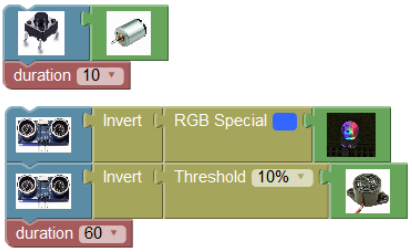

Fig. 5. Two sets of rules that implement a small control system for an automatic door and its alarm.

It is evident from the examples that with LAWRIS, projects of varying complexity can be implemented. The young student can begin with the simplest programs and continue to the more complex ones, while learning to program physical devices.

V. Conclusions

This paper presented LAWRIS, a programming system for physical computing intended for young students. The system is web-based and controls a specific Arduino circuit. LAWRIS is based on rules that young students may find easier to understand and implement. It features fast responses to both circuit input changes and programming modifications, making tinkering easier for beginner programmers. Since the system is based on Javascript (Blockly) and PHP, and the main logic is implemented in the Arduino, there is no need for real time processing on the host. Therefore, LAWRIS can operate on affordable computers that feature Arduino connectivity. The accompanying circuit is low-cost because it consists of cheap mainstream devices. Implementing simple programs is really straightforward with the proposed system and students can move on to implement more complicated projects as they learn more about programming and physical computing.

References

[1] D. Kushner, "The making of arduino," *IEEE Spectrum*, 26, 2011.

[2] C.Kelleher, and R. Pausch, "Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers," *ACM Computing Surveys*, vol. 37, pp. 83-137, 2005.

[3] J. Maloney et al., "Scratch: a sneak preview," Int. Conf. on Creating, Connecting and Collaborating through Computing, Jan. 2004, pp. 104-109.

[4] Y. Kato, "Splish: a visual programming environment for Arduino to accelerate physical computing experiences," Int. Conf. on Creating, Connecting and Collaborating through Computing, Jan. 2010, pp. 3-10.

[5] J. da Silva Gillig, "MiniBloq," 2011. [Online]. Available: http://blog.minibloq.org/. Accessed: Feb. 16, 2016.

[6] A. Millner and E. Baafi, "Modkit: blending and extending approachable platforms for creating computer programs and interactive objects," Int. Conf. on Interaction Design and Children, June 2011, pp. 250-253.

[7] Citilab, "S4A," 2015. [Online]. Available: http://s4a.cat/. Accessed: Feb. 16, 2016.

[8] D. Touretzky, "How should young children approach programming?," *Journal of Computing Sciences in Colleges*, vol. 29, pp. 5-6, Oct. 2013.

[9] "Blockly | Google developers," Google Developers, 2015. [Online]. Available: https://developers.google.com/blockly/. Accessed: Feb. 16, 2016.

[10] M. Resnick and E. Rosenbaum, "Designing for tinkerability," *Design, Make, Play: Growing the next Generation of STEM Innovators*, pp. 163-181, 2013.

[11] H. Partovi, "A comprehensive effort to expand access and diversity in computer science," *ACM Inroads*, vol. 6, no. 3, pp. 67-72, Sept. 2015.

[12] E. Järvinen, A. Karsikas, and J. Hintikka, "Children as innovators in action – a study of microcontrollers in finnish comprehensive schools," *Journal of Technology Education*, vol. 18, no. 2, pp. 37-52, 2007.

[13] D. van Laar, "Evaluating a colour coding programming support tool, " In Conf. of the British Computer Society, 1989, pp. 217-230.