2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)

Turning Pololu 3Pi into a multi-programming platform

José Roberto Fonseca S. Jr.
Maracatronics
Federal University of Pernambuco
Recife, Brazil
jrfsj@cin.ufpe.br

Pedro Vitor S. G. de Lima
Maracatronics
Federal University of Pernambuco
Recife, Brazil
p.vitor31@hotmail.com

Maria Helena R. A. Bezerra
Maracatronics
Federal University of Pernambuco
Recife, Brazil
m.helenalencar@gmail.com

João Marcelo X. N. Teixeira
Department of Electronics and Systems
Federal University of Pernambuco
Recife, Brazil
jmxnt@cin.ufpe.br

João Paulo Cerquinho Cajueiro
Maracatronics
Federal University of Pernambuco
Recife, Brazil
joaopaulo@ufpe.br

***Abstract*—Line-following robots have the ability to recognize and follow a line drawn on a surface. It works based on a simple self-sustainable system composed with a set of sensors, motors and a controller. In order to get optimal performance in such robots, it's necessary to carry out several tests to evaluate the behavior in each trial. In the majority of cases, a new trial requires to upload a new program, thus slowing down the development of the line-following. This paper presents an approach to solve the inconvenience of having to upload a new program in each trial. It consists in merging multiple codes in to one to create a program that gives the user the ability to switch between them anytime inside *Pololu's 3pi* line follower platform.**

***Index Terms*—multi-programming, line-following robot, autonomous robot**

I. Introduction

Line-following robots are autonomous robots based on a set of sensors, motors, and a controller designed to follow a predetermined line. One of the most common applications of line-following robots are in the area of education due to its ease of use while allowing the introduction of important microcontroller concepts, as analog inputs, outputs with Pulse Width Modulation (PWM), and decision making, besides facilitating the teaching of mathematics, physics and the use of the STEM methodology (a curriculum based on the idea of educating students in four specific disciplines – science, technology, engineering and mathematics) in the elementary education.Ve

The sensors usually chosen for detecting the line are Infrared (IR) reflective sensors [1], which can distinguish a white line from a black surface – or the opposite. The data from these sensors are continuously obtained by a controller and used to estimate the position of the robot above the line. The velocity and direction provided to the actuator (wheel motors, in the case of a differential robot [2]) are based on this estimated position. The robot should keep following the line track without deviating much from it by executing this measure–estimate–actuate cycle [3].

The process of developing an optimal line-following program for a given a track, i.e. performing the smallest lap time possible consists in firstly experimenting different logical approaches and then testing slightly variations of the same program – in order to find the best parameters. This process may take several hours and, considering that in every trial it's necessary to upload a new program, dozens of uploads are made to the controller. Thus a lot of the development time is spent on the inconvenient task of uploading programs to the robot.

It is also relevant to note that in several situations it is interesting to upload programs other than line-following ones, such as calibration routines (to determine the best threshold for the sensors) and tests.

Considering the classroom scenario, especially in elementary school where the time of each class is short, often the teacher uses more than one functionality of the robot and for each one to be tested, in a simple way, it is necessary more than one code. Thus another problem arises at the educational level, which is the unproductive time of the class due to the compilation of several codes.

Some robotic platforms, like the LEGO Mindstorms EV3 [4], have the ability to record several programs in its memory, keeping track of previously uploaded programs as a backup and also giving the user the capability to upload multiple programs at once. This controller has an LCD screen, buttons and a graphical user interface, so the user can interactively choose what program to run inside LEGO's platform. More *bare-bones* platforms, like Arduino boards [5] or Pololu 3pi [6], doesn't have that feature.

The Pololu 3pi can be programmed using the Arduino IDE and language [7], a feature that facilitates its use as an educational entry-level robotics platform. Given that the Pololu 3pi robot already has buttons and a LCD screen to serve as an human-machine interface (HMI), it was chosen as the platform to implement the multi-program feature.

978-1-5386-7761-2/18/$31.00 ©2018 IEEE
DOI 10.1109/LARS/SBR/WRE.2018.00103

This paper presents an online tool that enables loading multiple independent Arduino programs on the Pololu 3pi platform. The selection of which program will run is made at start-up, by its buttons and LCD screen.

The organization of this paper is as follows: Section II reviews existing educational robotic platforms. Section III describes the tool that enables loading multiple programs in the Pololu 3pi, its use, and implementation. Section IV discusses the impact of this tool in the memory usage of the platform and in the time needed for uploading various programs with and without the proposed tool, as well as known limitations of the tool. Section VI concludes the paper, resuming the results and describing future venues for this work.

II. Educational robotic platforms

According to [8], educational robots have become a significant asset to improve students critical thinking and creativity. The most popular platforms allow both constructing and programming the robots, while others only allow the provided robots to be programmed as they are. Some popular platforms are listed as follows and shown in Figure 1.

The LEGO Mindstorms EV3 [4] is one of the most used platforms not only because it can easily be structured but also because it allows multi-faceted use [4]. The robot's main unit is called "programmable brick" and can be connected to different sensors and actuators. LEGO also provides the "LEGO Mindstorms EV3 Programming", based on LabView, which allows programming using blocks. Third parties software allows code-based programming, using C, Java and others languages.

VEX Robotics is a robotics platform created by NASA engineers with the purpose to stimulate students scientific elements developed in the classroom through the concept of engineering prototyping [9]. It can be considered an alternative to the LEGO Mindstorms platform as it is also based on a main processing unit (block) connected to sensors and actuators.

Both platforms aforementioned are the basis used in the two biggest STEM competition in the world [10]–[12], are distributed in more than 35 countries and used by more than 500,000 students and teachers in local, regional and national competitions.

They also allow constructing both hardware and software. The last one only allows programming of the provided robots, while the next two platforms are in between the five platforms listed in this section, in which they are complete systems, but are capable of expansion.

The BBC micro bit [13] is a platform created as an alternative to Arduino that eases programming effort by enabling native programming by blocks, Javascript or Python. Different kits developed for this platform, such as the :MOVE mini buggy kit from Kitronik [14], are capable of transforming the single board into a complete robotic platform using wheels attached to servo motors for locomotion and an embedded power supply unit using batteries. One aspect of this platform is that it doesn't have line sensors, although it is possible to add more hardware, including line sensors.

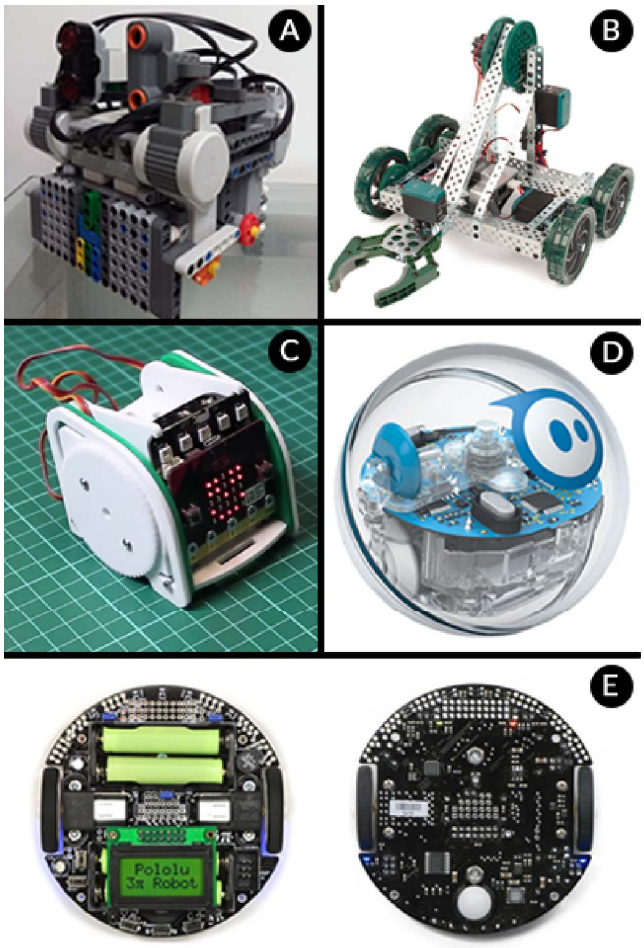

Fig. 1. Educational robotic platforms examples: A) Lego Mindstorms EV3; b) VEX Robotics; c) BBC micro bit with ::MOVE mini; d) Sphero SPRK+; e) Pololu's 3pi.

The Sphero Edu platform [15] is an educational solution provided by Sphero, a robot producer company specialized on the creation of spherical robots and their control. The platform enables programming the robots in three distinct levels: by drawing a path for the robot to follow, by blocks or by text using Javascript.

The Sphero Edu app runs on mobile devices and the upload of the program to the robot is done via Bluetooth. Since it is a platform with few or almost no accessories for the robots, it cost about four times less compared to LEGO and VEX solutions. The sensors available on the majority of Sphero robots are accelerometer, gyroscope and motor encoders. Because of that, they are not feasible to be used for autonomous control of line-following scenarios.

Pololu's 3pi platform was specifically developed for line-following activities. Without accessories, this robot has two micro metal gear motors, five IR reflectance sensors, an 8x2 character LCD, a buzzer, and three push buttons, which work as input interface with the user. It features a boost voltage

regulator that runs the motors at a constant 9.25 V independent of the battery charge level, which allows the robot to reach speeds up to 1 m/s and make precise turns [6]. This robot has a differential steering system, i.e., the direction of movement is given through the electronic control of the motors, so there is no need for an additional mechanics to control the direction of the robot. The 3pi's accessories transform it into a robot with many other features like wall follower, obstacle avoidance, line maze solver and even allow to turn it into a radio controlled robot.

The work proposed in this paper has as goal to bring the ability to upload different programs to the robotic platform and running the chosen one according to the user input – such as in LEGO and VEX platforms – to Pololu's 3pi. Next section describes how this goal was achieved.

III. The proposed tool

The proposed tool, called 3piMulti [16], accepts multiple Arduino codes and joins them in a single Arduino file, that can be compiled by the Arduino IDE.

Any Arduino Sketch-based code can be interpreted as a C++ code with many predefined functions (such as `digitalRead()`, `analogWrite()`) and constants (such as `LOW`, `HIGH`, `INPUT`, etc.). The Arduino code must have the definition of 2 functions: `void setup()` and `void loop()`. The `setup` function executes a single time at the startup or reset of the system and then the `loop` function executes in an endless loop, as shown on Figure 2.

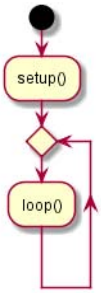

Fig. 2. Simplified execution sequence of an Arduino program.

Besides the `setup` and `loop` functions, it is possible to have in the code many other user-defined or library-defined functions and variables.

The 3piMulti tool joins several Arduino codes (`.ino` files) and generates a menu that allows the user to select from these different programs. Figure 3 shows an example of the intended sequence for three distinct input codes.

Since the proposed tool must "merge" multiple codes into a single `.ino` file, the generated code must also have `setup` and `loop` functions defined. The solution found was to assign the loop function to the one chosen in the menu and execute the equivalent setup function in the new global setup, as shown on Figure 4. This was performed by creating two void function pointers called `mainSetup` and `mainLoop`. The generated

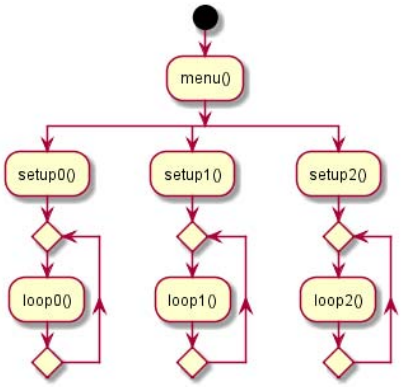

Fig. 3. Simplified sequence of a program that allows choosing from three distinct Arduino codes.

`setup` function sets the value for these two pointers based on the selection performed by the user. This way, whenever the generated code invokes the two function pointers, it is referring to the desired `setup` and `loop` functions.

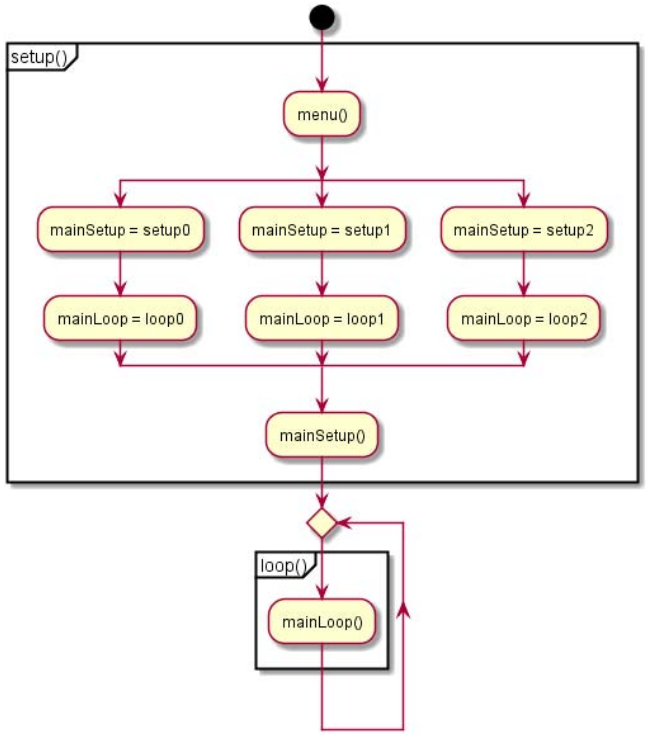

Fig. 4. Simplified execution sequence of a code generated from three distinct Arduino codes.

The tool parses each input code to identify variable and function names. All names that are local to the file are appended by an index. Both `setup` and `loop` functions are also suffixed. Once these modifications are done, the contents of each file are appended to the final code, together with the

code for generating the selection menu and the general `setup` and `loop` functions, as shown on Listing 1.

Fig. 5 compares the proposed selection menu on Pololu's 3Pi against the one from LEGO Mindstorms EV3. It also illustrates what the user should see in case he/she implemented a code similar to the one shown on Listing 1.

To recognize which identifiers are local and which are defined by the Arduino or by libraries used on the Pololu 3pi, the tool extracts a list of reserved words from the `keywords.txt` files used by the Arduino IDE for syntax highlighting [17]. The current version of the proposed tool considers only keywords defined by the Arduino IDE and those provided by Pololu 3pi libraries [18]. When parsing input files, comment sections (parts of the code starting with "//" or between "/*" and "*/") and string contents (parts of code between double quotation marks) are ignored, in other words, no suffix is appended to these sections.

Listing 1. Skeleton of the generated code for three sample input files.

```
/*------ MERGED CODE ------*/
void setup0() {/*...*/} // Prog. 1 setup code
void loop0() {/*...*/}  // Prog. 1 loop code
void setup1() {/*...*/} // Prog. 2 setup code
void loop1() {/*...*/}  // Prog. 2 loop code
void setup2() {/*...*/} // Prog. 3 setup code
void loop2() {/*...*/}  // Prog. 3 loop code
/*------ MENU ------*/
void(*mainsetup)();
void(*mainloop)();
void setup() {
  int option = 0;
  bool selected = false;
  while (!selected) {
    OrangutanLCD::clear();
    OrangutanLCD::gotoXY(0, 0);
    OrangutanLCD::print("Prog. ");
    OrangutanLCD::print(option + 1);
    OrangutanLCD::gotoXY(0, 1);
    switch (option) {
      case 0: OrangutanLCD::print("AnalogEx"); break;
      case 1: OrangutanLCD::print("BuzzerEx"); break;
      case 2: OrangutanLCD::print("LCDExamp"); break;
      default: break;
    }
    if(OrangutanPushbuttons::isPressed(BUTTON_A)) {
      OrangutanPushbuttons::waitForRelease(BUTTON_A);
      option = (option - 1) % 3;
    } else if(OrangutanPushbuttons::isPressed(BUTTON_B)) {
      OrangutanPushbuttons::waitForRelease(BUTTON_B);
      option = (option + 1) % 3;
    } else if(OrangutanPushbuttons::isPressed(BUTTON_C)) {
      OrangutanPushbuttons::waitForRelease(BUTTON_C);
      switch (option) {
        case 0: mainsetup = &setup0; mainloop = &loop0;
          break;
        case 1: mainsetup = &setup1; mainloop = &loop1;
          break;
        case 2: mainsetup = &setup2; mainloop = &loop2;
          break;
        default: break;
      }
      selected = true;
    }
    delay(100);
  }
  mainsetup();
}
void loop() {
  mainloop();
}
```

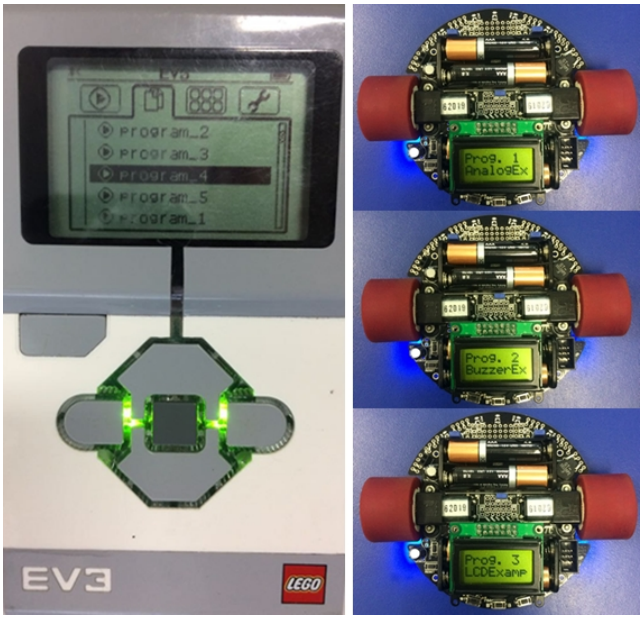

Fig. 5. Selection menus from LEGO Mindstorms EV3 (left) and the proposed solution (right).

IV. Analysis

To test the memory footprint of our tool, we chose a collection of 14 different programs, all of them being examples provided by Pololu's 3Pi library for Arduino IDE. A brief description of each tested program is given as follows:

1) **OrangutanAnalogExample**: reads the voltage output of a trimmer potentiometer (trimpot) in the background while the rest of the main loop executes. The LED is flashed so that its brightness appears proportional to the trimpot position.
2) **OrangutanAnalogExample2**: reads the voltage output of the trimpot (in millivolts) and also reads the Orangutan LV-168's temperature sensor.
3) **OrangutanBuzzerExample**: plays a series of notes on the buzzer.
4) **OrangutanBuzzerExample2**: plays a series of notes on the buzzer entirely in the background.
5) **OrangutanBuzzerExample3**: plays a series of notes on the buzzer allowing three distinct play modes to be selected.
6) **OrangutanLCDExample**: writes "Hello" on the LCD and then moves it around the display area.
7) **OrangutanLCDExample2**: displays custom characters on the LCD.
8) **OrangutanLEDExample**: controls the red and green LEDs on the 3pi robot.
9) **OrangutanMotorExample**: drives motors in response to the position of trimpot and blinks the red user LED at a rate determined by the trimmer potentiometer position.
10) **OrangutanMotorExample2**: drives motors in response to the position of trimpot and displays the potentiometer

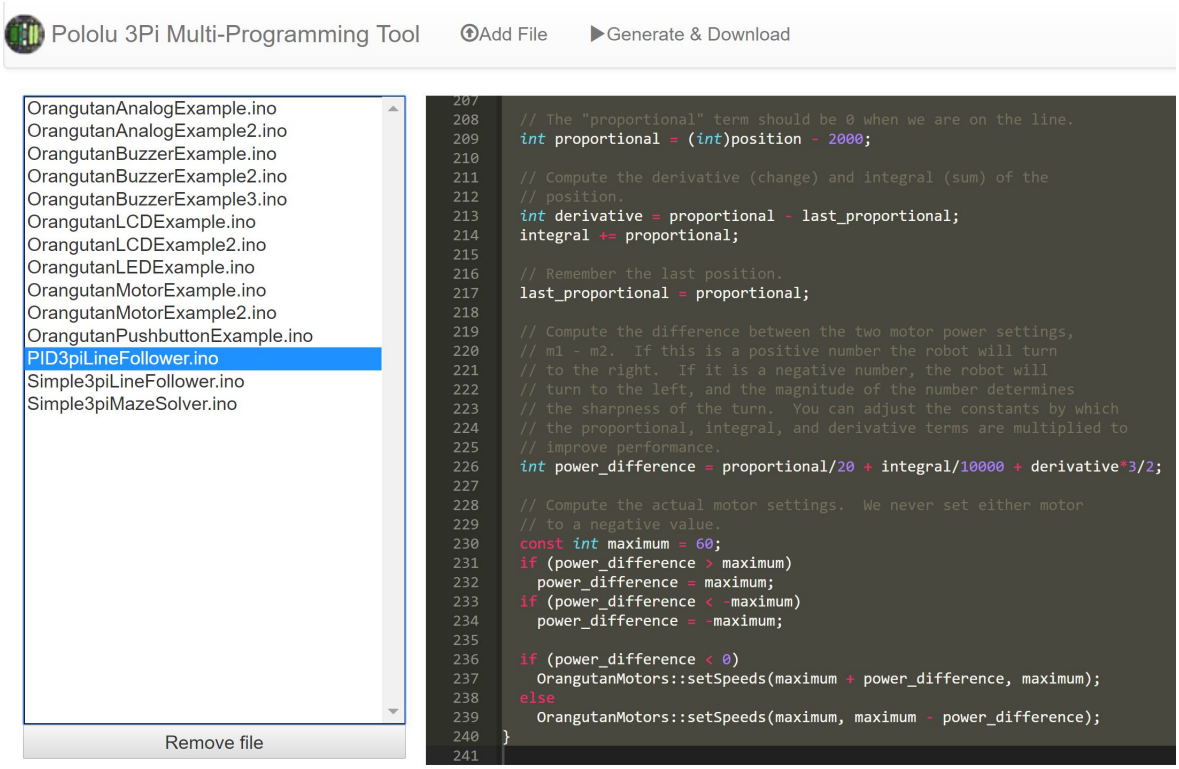

Fig. 6. Screenshot of the developed tool: the toolbar above has enables the user to upload multiple files and than download the merged code. The left container shows the uploaded files and the right container shows the final merged code (which the user can edit inside the tool)

position and desired motor speed on the LCD.

11) **OrangutanPushbuttonExample**: detects user input from the pushbuttons, and displays feedback on the LCD.
12) **PID3piLineFollower**: follows a black line on a white background using a PID-based algorithm.
13) **Simple3piLineFollower**: follows a black line on a white background using a very simple algorithm.
14) **Simple3piMazeSolver**: solves a line maze constructed with a black line on a white background, as long as there are no loops.

Each program was compiled separately in the Arduino IDE, as well as the merged code generated by 3piMulti (`3piMulti.ino`). All of them were successfully compiled and Table I shows the amount of memory used by each one after compilation.

One interesting point to observe is that the last 3 programs (the most memory demanding ones), `PID3piLineFollower`, `Simple3piLineFollower` and `Simple3piMazeSolver`, require at least 22% of the available memory, each. When compiled together, after passing through the proposed tool, all 14 codes require only 45% of the available memory. This happens due to the high memory footprint demanded by any Arduino sketch. Since all codes share the same base code (initialization, setup and loop structure, etc), only one instance is needed, optimizing memory use. Summing the number of bytes needed by each program being compiled individually, the total gives 47604 bytes, against 14832 bytes for the merged code generated. This means a reduction, consequently an improvement, of 68,8% in memory consumption, with no harm to the program's performance.

TABLE I
MEMORY FOOTPRINT OF 14 SEPARATE PROGRAMS AND THE MERGE OF THESE PROGRAMS (VALUES MEASURED IN BYTES AND RELATIVE PERCENTAGE IS GIVEN INSIDE PARENTHESES).

Program	Program Memory	Dynamic Memory
`OrangutanAnalogExample`	724 (2%)	14 (0%)
`OrangutanAnalogExample2`	1,912 (5%)	26 (1%)
`OrangutanBuzzerExample`	3,698 (11%)	357 (17%)
`OrangutanBuzzerExample2`	3,948 (12%)	146 (7%)
`OrangutanBuzzerExample3`	3,736 (1%)	87 (4%)
`OrangutanLCDExample`	1,276 (3%)	22 (1%)
`OrangutanLCDExample2`	1,964 (5%)	31 (1%)
`OrangutanLEDExample`	738 (2%)	9 (0%)
`OrangutanPushbuttonExample`	1,432 (4%)	51 (2%)
`OrangutanMotorExample`	1,208 (3%)	10 (0%)
`OrangutanMotorExample2`	2,108 (6%)	31 (1%)
`OrangutanPushbuttonExample`	1,432 (4%)	51 (2%)
`PID3piLineFollower`	7,582 (23%)	187 (9%)
`Simple3piLineFollower`	7,434 (22%)	181 (8%)
`Simple3piMazeSolver`	8,412 (25%)	1296 (14%)
`3piMulti`	14,832 (45%)	1,189 (58%)

Another test was made regarding productivity enhancement

due to the proposed tool's use. The total time necessary to individually compile each of the 14 programs and then upload them to the robot was measured (first approach). Comparatively, the total time necessary for using the proposed tool to generate the merged code, compile it and upload it to the robot was also measured (second approach).

The time needed for selecting each of the 14 programs on the robot was also included in this total time. While the first approach was concluded in approximately 570 seconds, the second one needed only 80 seconds, which means that there is a clear gain in productivity when the proposed tool is used, reducing the unproductive time during class caused by the delay in compiling the codes.

V. Known limitations

Since the proposed tool is in its first version, some limitations were already identified and are listed as follows.

The tool does not allow using any reserved word either from Arduino or from Pololu libraries, even if the use would be as a local variable or a member of a user-defined class. This happens even if the library that defines the specific reserved word is not used.

For the moment, it is not possible to revert the process, which means that the multiprogramming code generated by the tool cannot be automatically converted back to its original input files.

Another barrier noticed was that, unlike the LEGO Mindstorms, the tool developed cannot make the 3pi keep track of previously uploaded programs. This feature was not possible to achieve given that the tool developed *virtually* makes the Pololu's capable of storing more than one program. The platform still stores only one program at once, erasing the current loaded each time a new program is being uploaded.

VI. Conclusion

This work proposed a web-based tool capable of combining multiple codes for the Pololu's 3pi platform and generating a single file, mimicking the program selection functionality from platforms such as LEGO Mindstorms and Vex Robotics. A clear advantage of such tool is the decrease in testing time, which is very important in line-following competitions, given the limited time available to make tests using the real competition arena, and in class, increasing the time available and allowing the teacher to perform other activities or explanations. By merging different programs together, we showed that it was possible to make better use of 3pi's program memory, in comparison to using each program alone. Using the proposed tool also resulted in productivity improvements.

Another relevant gain is the possibility of exporting the concept and logic applied to this tool to Arduino and Arduino-compatible platforms.

As future work directions, we intend to support saving the variables values on Pololu's 3pi non-volatile memory, so that by restarting the platform some values are not lost.

We also intend to allow sharing variables between different programs. This capability would bring two direct benefits: an optimized memory use, increasing the maximum number of programs that can be loaded into Pololu's 3pi at the same time; communication between different programs (a program could tune or set a constant to be used by another one, for example).

In a near future, besides generating the .ino source code for the merged programs, we intend to generate the binary files for direct upload to the platform, performing a server-side compilation. This should come together with a self-contained tool for sending the downloaded binary to the 3pi. At last, we intend to use more sophisticated approaches such as JSCPP [19], which is a C++ interpreter written in Javascript. This would allow a more intelligent combination of the input programs, solving the variable/function naming problem presented in the Analysis Section.

Acknowledgments

We would like to thank Pololu in the names of Ben and Claire for supporting our research and giving us an incredible tour at Pololu's headquarters.

References

[1] Pololu. Qtr-1a reflectance sensor. [Online]. Available: https://www.pololu.com/product/2458, month = jul, year = 2018
[2] R. SIEGWART and I. R. NOURBAKHSH, *Introduction to Autonomous Mobile Robots*. A Bradford Book, 2004.
[3] K. Z. Haigh and M. M. Veloso, "Planning, execution and learning in a robotic agent." in *AIPS*, 1998, pp. 120–127.
[4] İ. KUNDURACIOĞLU, "Examining the interface of lego mindstorms ev3 robot programming," *Online Learning*, vol. 1, no. 1, 2018.
[5] M. Margolis, *Arduino Cookbook*. O'Reilly Media, Inc., 2011.
[6] 3pi. (2018, Jul.) Pololu 3pi robot. [Online]. Available: http://www.pololu.com/product/975
[7] M. Banzi and M. Shiloh, *Getting started with Arduino: the open source electronics prototyping platform*. Maker Media, Inc., 2014.
[8] T. B. Brahim, D. Marghitu, and J. Weaver, "A survey on robotic educational platforms for k-12," in *E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*. Association for the Advancement of Computing in Education (AACE), 2012, pp. 41–48.
[9] C. C. Hendricks, M. Alemdar, and T. Ogletree, "The impact of participation in vex robotics competition on middle and high school students' interest in pursuing stem studies and stem-related careers," in *American society for engineering education*. American Society for Engineering Education, 2012.
[10] Lego. (2018, Jul.) First lego league. [Online]. Available: http://www.firstlegoleague.org/
[11] V. Robotics. (2018, Jul.) Vex robotics competition. [Online]. Available: http://www.vexrobotics.com/competition
[12] R. Federation. (2018, aug) Robocup. [Online]. Available: http://www.robocup.org/
[13] A. Schmidt, "Increasing computer literacy with the bbc micro: bit," *IEEE Pervasive Computing*, vol. 15, no. 2, pp. 5–7, 2016.
[14] Kitronik. (2018, aug) Kitronik. [Online]. Available: http://www.kitronik.co.uk/5624-move-mini-buggy-kit-excl-microbit.html
[15] S. M. Hadfield, M. G. Hadfield, M. D. Sievers, and J. T. Raynor, "Experience hands-on k-12 and first-year university coding curricula based on the sphero sprk+ robot," in *Proceedings of the 23rd Western Canadian Conference on Computing Education*. ACM, 2018, p. 18.
[16] Maracatronics. (2018, aug) Pololu 3pi multi-programming tool. [Online]. Available: https://maracatronics.github.io/3piMulti/
[17] Arduino. (2018, Jul.) Writing a library for arduino. [Online]. Available: http://www.arduino.cc/en/Hacking/libraryTutorial
[18] Pololu. (2018, aug) Configuring the arduino environment. [Online]. Available: http://www.pololu.com/docs/0J17/3

[19] F. Hao. (2018, Jul.) Jscpp: a simple c++ interpreter written in javascript. [Online]. Available: https://github.com/felixhao28/JSCPP