Instruction of Introductory Programming Course using Multiple Contexts

David W. Brown
Tennessee Technological University
Cookeville, Tennessee, USA
dwbrown@tntech.edu

Sheikh K. Ghafoor
Tennessee Technological University
Cookeville, Tennessee, USA
sghafoor@tntech.edu

Stephen Canfield
Tennessee Technological University
Cookeville, Tennessee, USA
scanfield@tntech.edu

ABSTRACT

This paper describes the experience of redesigning a traditional CS1 programming course, utilizing traditional coding practices as well as microcontroller units (MCU) based coding, to provide multiple programming environments. The objective of this redesign is to improve the programming skills for engineering students by 1) providing them with program development experience in multiple contexts and 2) relating the initial programming experience to the typical notion of engineering through significant hardware experience. Typical CS1 courses are designed with an instructor led lecture focusing on the introduction of specific computer skills and languages while programming assignments and laboratories help strengthen these skills in the students. For this remodeling, in addition to the typical programming exercises, supplementary MCU based lab exercises were used to provide an additional, different programming target for increased learning and highlighting the complementary relationship between hardware and software. The outcomes of this effort demonstrate that the addition of a MCU to an introductory programming course can work as an effective motivator, providing the students with a secondary context to reinforce programming skills developed during the course, and that providing multiple contexts (traditional desktop programming and hardware-based programming) together can aid in learning and the transfer of knowledge.

CCS CONCEPTS

• **Social and professional topics** → **CS1**; **Computer engineering education**; • **Hardware** → *Integrated circuits*; Printed circuit boards;

KEYWORDS

Microcontroller, Multiple Contexts, Knowledge Transfer, Introductory Programming

ACM Reference Format:
David W. Brown, Sheikh K. Ghafoor, and Stephen Canfield. 2018. Instruction of Introductory Programming Course using Multiple Contexts. In *Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3197091.3197105

ITiCSE'18, July 2–4, 2018, Larnaca, Cyprus

ACM ISBN 978-1-4503-5707-4/18/07…$15.00
https://doi.org/10.1145/3197091.3197105

1 INTRODUCTION

Numerous research studies have established that learning introductory computer programming is difficult for incoming freshmen [3, 16]. Combine what is traditionally a difficult course, with a section of students who do not comprehend why the subject is required in their discipline and the course becomes much more problematic. The traditional entry-level programming course for engineers is based on learning programming using C/C++, Java, FORTRAN or Matlab to solve numerical algorithms associated with common engineering models. Any use of a computer as a device to control physical events is generally reserved for upper level courses. While creating programs to solve numerical analysis problems is an important tool for engineers, we argue that the current model is inverted for engineering students on a pedagogical basis. Ideally, students would begin learning programming in an environment that would match the notion of what engineers do, namely that they design systems that control the world around them, and then later move to solving advanced models that describe how the world works. The majority of the population has a belief that engineers 'build things' and that makes it harder to comprehend why something they cannot touch is important [15]. In school, we teach basic, fundamental skillsets, and then ask the student to apply these skills to solve problems with different data or in a different context in later classes or as a member of the workforce. In our work we redesigned a traditional CS1 class that supports the student's mental model of engineering. To accomplish this, we looked at providing more than a single context for learning programming. Although others have used similar departures from the standard instructional environment (e.g. MATLAB, exotic hardware, etc.), it has been a focus of ours from the beginning to attempt to bridge the gap between misconception and reality to tap into the engineering students' natural passion. While most of the existing research has involved moving entirely to a different programming context, we chose to do this by using the MCU as a secondary contextual device rather than as the sole programming reference. In our redesigned course, students are required to program a MCU to control hardware such as LEDs, speakers, servo-motors, etc. in the lab portion of class in addition to typical PC-based programming assignments. We believe that the redesigned course provides a learning environment early in the curriculum which still matches the students' notion of engineering. In addition, the course provides more than one context of learning which we believe will aid in the better transfer and retention of programming knowledge and skills.

2 RELATED WORKS

Improvements to the pedagogy of undergraduate engineering programming instruction have attracted significant attention in literature to date. Since most engineering instruction in programming skills takes place in the freshman year, it is natural that freshmen are often the focus for research into instructional technique improvements [1, 2, 4]. Numerous research projects have been carried out to investigate the use of robotics or MCU programming as a means of teaching non-Computer Science majors Beginning with Schumacher and Fagin's work [8, 13] in implementing introductory computing courses using LEGO MindStorm and continuing through the works of Xu and Panadero, [11, 17] there has been a long history of replacing portions of a laboratory curriculum with assignments based on robotics systems. In addition, a common theme has been the use of non-traditional programming assignments such as spreadsheets [5], MATLAB [12] or flowchart applications [6] in place of languages such as Python, C or C++. It is only recently that studies have tried using simpler MCUs in these environments [9, 14]. One aspect of engineering learning that seems to have received little attention is the impact of students' (especially freshmen) misconceptions about what engineering is, and the sort of work that is done by engineers on a daily basis [7]. In addition, research shows that an issue exists of motivating students who feel that programming is a chore or a skill they are not likely to use in their chosen careers [15]. Quite often this introductory programming course represents these students first exposure to engineering which further alienates the students from what they believe engineers actually work on. Often, this misunderstanding is fed by a misconception held by society in general [10].

3 COURSE DESCRIPTION

Our course is designed as a traditional, semester-long, four credit hour CS1 course, with a three-hour lecture component and a one hour lab. As this is a course for engineers, the primary language used for programming in the class is C++. The lectures follow a traditional CS1 syllabus with lectures, weekly quizzes, programming assignments, and tests. Table 1 shows the topics covered during the lecture portion of the class. In both the lecture and for portions of the lab, the target for programming examples and assignments is the traditional desktop PC. However, each lab assignment also includes an assignment that mirrors the main topic being covered but are designed for the MCU. This MCU programming provides an additional context that is appropriate for the demonstration of engineering practice.

Our university is a medium sized, accredited public engineering university with an enrollment of approximately twelve thousand students. The college of engineering is the largest school on campus with an enrollment of over three thousand students. The freshmen students in Electrical Engineering, Computer Engineering, Computer Science, Mechatronics and Manufacturing Engineering Technology are required to take CS1 as part of their curriculum. This CS1 course is taught by the Computer Science Department regardless of the student's individual major. No prior programming experience is required to take this course, and in fact, most of the students do not have prior experience. The enrollment for this class has continued to grow over the past 10 years and now every semester more than two hundred students take this course. The course is divided into two lecture sections and six to seven lab sections, consisting of 25-30 students. The redesigned class was implemented in two phases. The first phase was implemented during the 2011-2012 academic year. During that time, additional sections of lecture were taught providing a lecture and lab class size of approximately 30 students. The second phase has been started in fall 2015 and is continuing.

Table 1: Course Topics

Topic Num	Topic
1	Programming Design Process
2	Variables, Data Types and Math Expressions
3	Input and Output
4	Conditional Statements (if, if/else, switch)
5	Looping (for, while, do...while)
6	User Defined Functions
7	Arrays (one and multi-dimensional)
8	Pointers
9	Structures
10	Debugging and Error Correction

3.1 Course Hardware

A primary distinguishing feature of this course is that it provides a hardware-based context, in addition to a PC context, as the initial programming target for the lab portion of the class. In the first phase, the MCU in question was a Dragon12 Plus board. This board was chosen for the initial evaluation period for several reasons. At the time, it was the board used for upper division mechatronics, embedded systems courses, as well as for many senior level design projects in the electrical and Computer Engineering department. The second reason for choosing the Dragon12 board was that project was funded by National Science foundation and the grant proposal which funded the first phase of this research was written for a redesign effort with this board. The boards in question were supplied to the students for the duration of the class, then returned to the department for use in subsequent semesters. In the second phase, the Dragon12 board was replaced with an Arduino-based board designed for introductory circuitry courses (Figure 1). Several reasons existed for us to undertake this change. Primarily, the difference was a cost improvement between the Arduino boards and the Dragon12 board. The difference is understandable when considering the differences between the two boards, the Dragon12 board is a complete fully functioning MCU while Arduino boards are cheaper but more expandable through add-on components. This cost savings further allows the department to provide a board to the students that they are able to keep, promoting experimentation after the class is completed. A secondary consideration is a movement on the part of the Electrical Engineering department away from the Dragon boards in their senior level coursework. The cost of the board is approximately 60 USD, which most of the students are willing to pay since the cost works out to be less than the cost of one standard textbook. The hardware setup for the second phase our experiment was an integrated Arduino MCU which contained an integrated temperature sensor, three separate potentiometers,

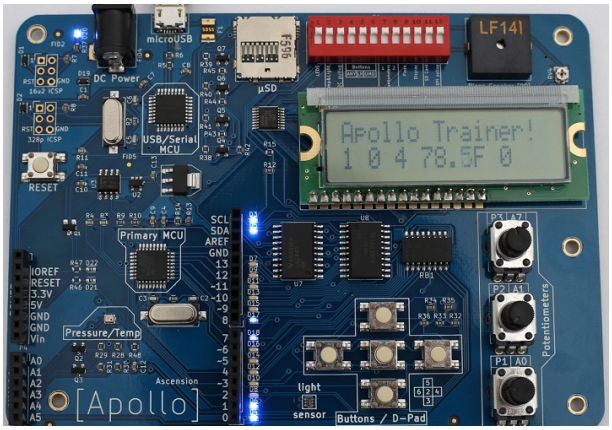

Figure 1: Arduino MCU Board

5 user controllable buttons to provide input, a 16 by 2-character LCD display for text output, light sensors, a piezoelectric speaker, a digital to analog converter, and the ability to transfer information from a computer to the card through either USB support or a micro SD card. Combined, these features allowed us to develop a wide variety of challenging laboratory exercises for the students, while also providing them enough features to encourage their continued experimentation. Changes to the MCU also required a change in the software environment the students used to program the laboratory assignments. While using the Dragon boards, the Freescale CodeWarrior cross compiler, which comes with an Integrated Development Environment (IDE), was used to allow students to compile a C/C++ program on their desktop and download the executable to the target Dragon 12 board through the USB connector in a single step. With the change to the Arduino, the standard Arduino Integrated Development Environment (IDE), was used to compile a C/C++ program on their desktop and download the executable to the target MCU through the USB connector in a single step. While the syntax for using this editor is C++-like and it allows the same functionality, some of the standard input and output functions are slightly different. As a result, rather than worrying about the low-level details of the hardware, students are able to spend more time in analyzing the problem and developing programs for the hands on activities.

3.2 Laboratory Assignments

Portions of weekly lab assignments (over a 14 week semester) based on the MCU were given to the students. Face to face instructor/TA contact time for the lab was 1.5 hours, however, if the student was unable to finish the assignment during this time, they could continue work on their own, for a period of 3 to 4 days. For the hardware portions of lab, students were able to work directly on the MCU boards, which were issued at the beginning of the semester. Table 2 gives examples of the lab assignments correlated with the programming constructs which were the learning objectives for each of the labs. The first lab involved student familiarization with the MCU as well as downloading and setup of the Arduino IDE and executing a simple "Hello World" type program on the MCU. Each lab activity was assigned along with a problem statement and a required set of deliverables. To fully complete each assignment, the student was required to implement their programmed device in a setting outside of the classroom or lab. An exact description of sample labs are presented below

3.2.1 Laboratory Assignment - Speed Game. This lab acts as the fifth lab involving the Arduino and involves making a simple speed game. The primary programming concepts the lab highlights are the use of loops to repeat code and conditional statements to control input and output. To accomplish this, the users build a simple speed game to be played by two people. The students program the piezoelectric speaker to sound and flash the LED lights to signify that a game is about to start. After this signal is given, a break of a random time between 1 and 10 seconds is taken. After the break, one of the LED lights flashes on until one of the two user buttons is pressed. The person who presses their button first is declared the winner. A tone sounds on the piezo speaker and the LED lights will flash for 100 milliseconds. In addition, the LCD display will show a message indicating that the game is over and indicating which player won. After a 4 second delay, the game is designed to be restarted. Loops control the delays, lighting of the different LCD and LED components and restarting the game, while conditional statements allow the game to indicate winner for each round. The students are encouraged to add their own touches to the game with a suggestion of keeping track of the number of wins for each player.

3.2.2 Laboratory Assignment - Lock Box. This lab acts as the tenth lab involving the Arduino and involves making a locking mechanism suitable for a safe. The primary programming concepts the lab highlights are the use of arrays to store related data, functions to create modular code, loops to allow repeated use of the same code and conditional statements to control input and output. To accomplish this lab, the students must code a number of functions including one for getting the original lock box password, one to check if a supplied code matches the original, a third for presenting a 'locked state' to the user and finally a function to handle what occurs if an incorrect password is entered. The initial user password is stored in an array and is made up of presses made to any of the five built in buttons present on the MCU. Once this password is entered, the MCU is considered locked and waits for a second user to attempt to unlock it. The MCU will stay locked until the correct sequence of buttons is entered which will call the unlock function. Once again, students are given the freedom to control what occurs when an incorrect, or correct, passcode is given using any of the components present on the MCU.

4 EVALUATION

In the first phase of our experiment, which occurred during the 2011–2012 academic year, students signed up for sections in the normal fashion, with all sections containing both CS and non-CS engineering students. During each semester, a single section of class consisting of 30 students was selected at random to act as an intervention group while a second was chosen as a control. The intervention section performed labs that were entirely based on Dragon12 boards while the control lab, though taught by the same

Table 2: Lab Exercises by Programming Construct

Concept	Task	Description
Input/Output	Students learn how to use a microcontroller to interact, through buttons, analog inputs and LED displays	Using potentiometers and the built in ADC on the Arduino students cause LED lights and piezo speakers to "blink" as input is modified.
Conditional Logic	Using if statements the students will control lighting of LEDs based on button presses	The order in which built in buttons are pressed is used to determine if and when built in LEDs light up.
Loops	Control on and off blinking of LED lights based on user input	LED lights are set to blink in successive order until a button is pressed which causes all three buttons to blink simultaneously until the button is released.
Nested Loops	Using multiple LED's a display and a temperature sensor we simulate an air conditioning sensor	One light is set as heating, the other as cooling. The display indicates the currently observed temperature which can then be manipulated through the user touching a sensor.
Arrays	Simulate locking and unlocking a safe	Prompt for a passcode to lock; turn motor to simulate locking. Ask for passcode to unlock; compare with saved passcode and simulate unlocking by turning motor if passcode is correct.
Structures	Create music box	Store individual notes as a structure of a pitch and duration. Then store an array of notes as a song that can be played through a built in piezoelectric speaker

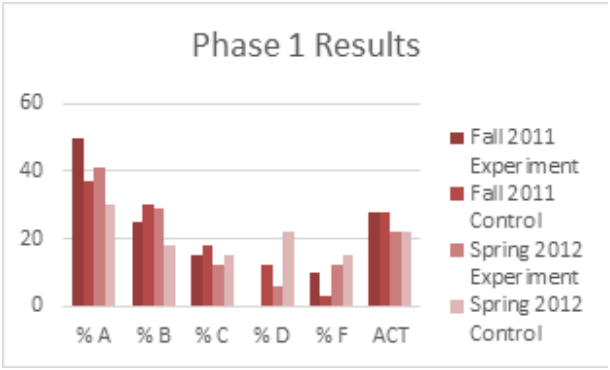

Figure 2: Preliminary Experiment Results 2011–12

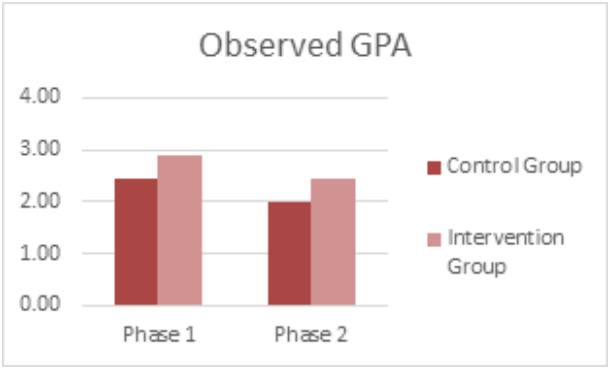

Figure 3: GPA Across Experiment

professors, used standard PC-based laboratory assignments. The final course grade containing the weighted average of quizzes, exams, programming assignments and laboratory assignments has been selected as the assessment indicator. As we can see, a much larger percentage of students were able to receive an A for the course from the intervention group during both the fall and spring semesters. (Figure 2). We can also see from the figure that the average ACT, or American College Testing, scores for both the intervention and control sections were equal. The ACT is an independent, standardized math, language, science, and reasoning test used for university admissions, and the scores would indicate that the two groups' background preparation for the class would be roughly equivalent. In addition, for the two semesters, the GPA for all students was noticeably higher, 2.93 in the intervention sections compared to 2.56 in the control sections (Figure 3). After examining the results from the first part of the trial, we felt the initial findings were promising but we also felt that the work needed to be more focused on the Electrical and Computer Engineering students for two reasons: 1) The faculty and leadership of the ECE Department wanted all EE and CE students to take an MCU-based CS1 class, because qualitatively they observed that ECE students who had taken the MCU course during phase 1 were move confident and were performing better in upper division MCU-based Mechatronics, or embedded systems classes compared to student who had not taken the MCU-based class. 2) There was criticism of the course from Computer Science students who did not want to take an MCU-based CS1 class stating that hardware was not part of their core curriculum. 3) It was a problem financially for the Computer Science department to purchase MCU boards for the CS students, while the ECE department was willing to invest in hardware for their students. With these changes in mind, we were able to work with leadership in the academic departments and the School of Engineering to create laboratory

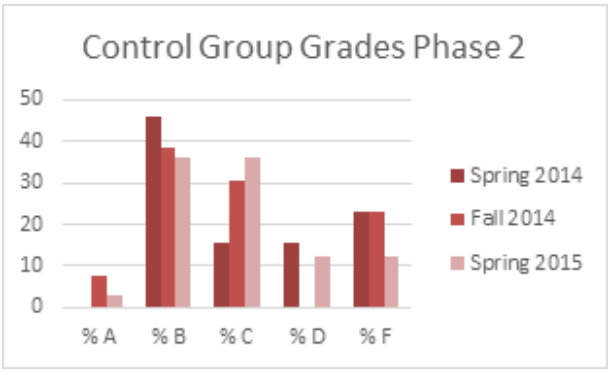

Figure 4: Grade Distribution Labs Without MCU

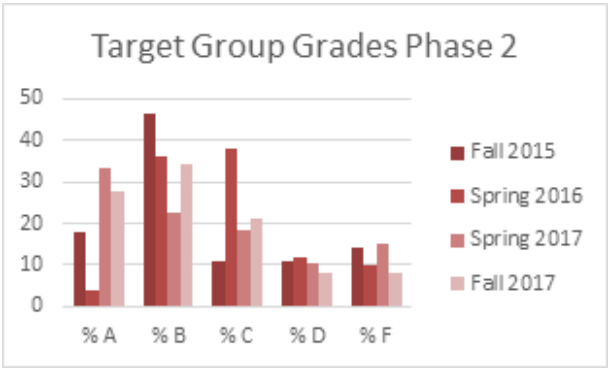

Figure 5: Grade Distribution Labs With MCU

sections that contained only non-Computer Science, mostly ECE, students. The lecture sections would continue to be mixed but all non-Computer Science students would be required to take specific lab sections. Since there was no control group for non-CS Engineering students in the second phase of our study, we have used grades of non-Computer Science students in the three semesters prior to the start of the second phase as our control group. Here we would like to mention that prior labs were of similar difficulty level as the labs constructed for these new sections. Figure 4 shows the percentage of each grade achieved by the control group in courses taught by one of the authors. A secondary development occurred as we were beginning phase two. Until 2016, lecture and lab sections had been limited in size to approximately 35 students. Beginning in the spring semester of that year, the lecture size was increased to 100 students, while multiple 30-35 student lab sections were held. This change in class structure meant that a larger portion of the one on one interaction would occur during labs, increasing their importance. Despite this change, to class size, students in the MCU based labs showed marked improvement (Figure 5) compared to the control group. In addition to the individual grade increases, as Figure 3 shows, the phase 2 GPA for the intervention group of 2.43, while not as high as those seen for phase 1 are still much higher than the 2.01 observed in the control group. Furthermore, anecdotal evidence in the form of feedback from the electrical and Computer Engineering department faculty indicate that they believe students in the intervention group are better prepared to succeed in the department after successful completion of the course.

5 LESSONS LEARNED

During the two-phase MCU-based CS1 redesign and implementation effort we have faced several challenges and learned the following lessons:

- Teaching CS1 to engineering students using MCU's in addition to traditional PC-based exercises is a viable model.
- During our first phase implementation and the first semester of our second phase implementation, students had to build some circuitry using wires, breadboards, and different sensors. Frequently because of a loose connection or due to incorrect wiring the program would not work as expected. This may become frustrating for students, especially as they are learning new material. From spring 2016, we have used a specialized Arduino board from a local vendor that contains all integrated circuitry. This has eliminated all circuitry or loose wiring related errors and frustrations.
- Additional work on the part of the professor may be necessary to effectively teach this course. The professor may need to learn at least some basics of electric circuitry. These extra responsibilities may require some additional motivation for traditional Computer Science professors.
- Extra lab support may need to be available to help students in these sections compared to traditional courses. As students start working in a MCU environment, they may initially grow frustrated. In our experience, this is more prevalent in sections using non-integrated circuitry where a misplaced wire can be the cause of errors rather than incorrect code. An additional aide in the course is therefore valuable to aid in encouraging the students and correcting these issues as soon as possible.
- Lab assistants or TAs should be either knowledgeable or have a knack or an interest in learning circuitry.

6 CONCLUSION

This paper has presented a model for the restructuring of the traditional CS1 Introduction to Programming course by adding a hardware-based lab component. The underlying pedagogical foundations of this activity are 1) to engage incoming students' notions of engineering and to build on this early knowledge in a progressive fashion with programming applications that are relevant to engineering and 2) to provide a hardware-based programming context in addition to traditional PC environment. The assignments are designed around the objectives of building on existing students' knowledge, enhancing knowledge transfer, and enabling students to take more control of their learning process. This redesigned programming course was implemented several times with a mixture of Electrical Engineering, Computer Engineering, Computer Science, Mechatronics and Manufacturing Engineering Technology students over multiple semesters. Assessments of the project provide a strong indication that the students engaged in the hands-on, hardware-based programming activities performed better their comparison group peers who were exposed only a single context using traditional PC assignments. Comments from the faculty of

the electrical and Computer Engineering department show that students in the intervention group are believed to be better prepared to succeed in the major after successfully completing the multiple context course. In conclusion, we contend that the results imply that the hands-on programming model provides increased engagement and builds on incoming notions of programming in engineering that result in better learning. The increased engagement appears to be due to the hardware based activities, while the better learning may in part stem from 1) increased engagement, 2) building on an existing framework of knowledge, and 3) exposure to programming in multiple contexts (both MCU hardware and desktop PCs).

ACKNOWLEDGMENTS

Portions of this research were funded by an NSF grant, DUE1044590.

REFERENCES

[1] Victor Adamchik and Ananda Gunawardena. 2005. Adaptive book: Teaching and learning environment for programming education. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, Vol. 1. IEEE, 488–492.

[2] John E Bean and John P Dempsey. 2007. Collaboration between Engineering Departments at Clarkson University for a Freshman-Level Engineering Programming Course Including an Experimental Lab Experience. In *CIEC 2007 Conference, 4p.*

[3] Jens Bennedsen and Michael E Caspersen. 2007. Failure rates in introductory programming. *AcM SIGcSE Bulletin* 39, 2 (2007), 32–36.

[4] David E Clough, Steven C Chapra, and Gary S Huvard. 2001. A change in approach to engineering computing for freshmen–similar directions at three dissimilar institutions. *age* 6 (2001), 1.

[5] Mauricio A Colombo, María Rosa Hernández, and Jorge E Gatica. 2000. Combining high-level programming languages and spreadsheets an alternative route for teaching process synthesis and design. *age* 5 (2000), 2.

[6] Thomas J Cortina. 2007. An introduction to computer science for non-majors using principles of computation. In *ACM SIGCSE Bulletin*, Vol. 39. ACM, 218–222.

[7] Michael Davis. 1998. Thinking like an engineer: Studies in the ethics of a profession. (1998).

[8] Barry S Fagin, Laurence D Merkle, and Thomas W Eggers. 2001. Teaching computer science with robotics using Ada/Mindstorms 2.0. In *ACM SIGAda Ada Letters*, Vol. 21. ACM, 73–78.

[9] Mark Goadrich. 2014. Incorporating tangible computing devices into CS1. *Journal of Computing Sciences in Colleges* 29, 5 (2014), 23–31.

[10] Louis S Nadelson and Janet Callahan. 2011. A comparison of two engineering outreach programs for adolescents. *Journal of STEM Education: Innovations and Research* 12, 1/2 (2011), 43.

[11] Carmen Fernández Panadero, Julio Villena Román, and Carlos Delgado Kloos. 2010. Impact of learning experiences using LEGO Mindstorms® in engineering courses. In *Education Engineering (EDUCON), 2010 IEEE*. IEEE, 503–512.

[12] Julián Ramos, María A Trenas, Eladio Gutiérrez, and Sergio Romero. 2013. E-assessment of Matlab assignments in Moodle: Application to an introductory programming course for engineers. *Computer Applications in Engineering Education* 21, 4 (2013), 728–736.

[13] Jerry Schumacher, Don Welch, and David Raymond. 2001. Teaching introductory programming, problem solving and information technology with robots at West Point. In *Frontiers in Education Conference, 2001. 31st Annual*, Vol. 2. IEEE, F1B–2.

[14] Hugh Smith. [n. d.]. Microcontroller Based Introduction to Computer Engineering. ([n. d.]).

[15] Lynda Thomas, Mark Ratcliffe, John Woodbury, and Emma Jarman. 2002. Learning styles and performance in the introductory programming sequence. In *ACM SIGCSE Bulletin*, Vol. 34. ACM, 33–37.

[16] Christopher Watson and Frederick WB Li. 2014. Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*. ACM, 39–44.

[17] Dianna Xu, Douglas Blank, and Deepak Kumar. 2008. Games, robots, and robot games: complementary contexts for introductory computing education. In *Proceedings of the 3rd international conference on Game development in computer science education*. ACM, 66–70.