Are There Differences in Learning Gains When Programming a Tangible Object or a Simulation?

Grégoire Fessard, Patrick Wang, Ilaria Renna

Are There Differences in Learning Gains When Programming a Tangible Object or a Simulation?

Grégoire Fessard
Institut Supérieur d'Électronique de Paris
Paris, France
gregoire.fessard@isep.fr

Patrick Wang
Institut Supérieur d'Électronique de Paris
Paris, France
patrick.wang@isep.fr

Ilaria Renna
Institut Supérieur d'Électronique de Paris
Paris, France
ilaria.renna@isep.fr

ABSTRACT

Physical computing is about programming and interacting with a tangible object to learn fundamental concepts of Computer Science (CS). This approach presents several benefits regarding motivation, creativity, and learning gains. Yet, these learning gains hardly are compared with those resulting from the programming of a simulation of a tangible object. In this article we present the results of a comparative study that has been conducted to explore this issue. With this study, we aimed to determine whether the programming of a tangible object or its digital simulation yields significantly different learning gains. In the experiment we conducted, participants (aged 14-17 with little or no prior programming knowledge) were divided into two groups: one programmed a tangible electronic board (the BBC micro:bit) while the other programmed a simulation of it. The results of this experiment suggest that, while each group significantly improved their understandings of fundamental CS concepts (i.e., variables, conditions, and loops), no significant difference was found when comparing the learning gains between the two groups.

CCS CONCEPTS

• **Human-centered computing → User studies**; • **Social and professional topics → Computing education**.

KEYWORDS

computer science education; comparative study; physical computing; digital computing; block-based programming

ACM Reference Format:
Grégoire Fessard, Patrick Wang, and Ilaria Renna. 2019. Are There Differences in Learning Gains When Programming a Tangible Object or a Simulation?. In *Innovation and Technology in Computer Science Education (ITiCSE '19), July 15–17, 2019, Aberdeen, Scotland UK*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3304221.3319747

ITiCSE '19, July 15–17, 2019, Aberdeen, Scotland UK

ACM ISBN 978-1-4503-6301-3/19/07…$15.00
https://doi.org/10.1145/3304221.3319747

1 INTRODUCTION

Physical computing refers to the "design of tangible and interactive object using programmable hardware" [19]. Physical computing differentiates itself from a more traditional teaching approach that uses the command line as sole interface to the program execution. With the development of programmable robots and electronic boards, researchers took an interest in physical computing as a means to introduce computer science and programming to teenagers [9, 24]. However, and while focused on identifying the benefits of physical computing, these studies do not compare their results with what could be achieved with *digital* computing (i.e., the programming of a simulation of a tangible object).

In this article, we present a study designed to determine whether physical computing is more beneficial than digital computing (or vice-versa) to the learning and understanding of basic programming concepts (namely variables, conditional structures, and iterative structures). For this study, we have conducted an experiment which uses the BBC micro:bit. This electronic board can be programmed using a block-based programming language and was provided to our participants under either its tangible or simulated form.

In the following section, we provide a literature review of the use of physical computing to introduce programming to teenagers. In particular, we also look at block-based programming languages as they are often implemented in physical computing systems. In Sections 3 and 4, we describe the programming environment and the experimental protocol used in this study. Section 5 presents the results obtained in our experiment. Finally, the last two sections are dedicated to a discussion of our results and a description of future works that we plan on conducting.

2 LITERATURE REVIEW

The literature review on Computer Science Education has thoroughly investigated the learners' misconceptions and difficulties related to programming. For example, du Boulay [5] identifies three main difficulties. The first one concerns *notation* which refers to the difficulty of mastering the syntax and semantics of a programming language. The second difficulty relates to *structures* i.e., designing algorithms with the smallest units of programming instructions. The third difficulty is understanding the concept of *notional machine* which means realizing that the behavior of a machine is dictated by the programs being executed and the interactions the user has with this machine. Some more practical difficulties are also mentioned in the state of the art, and in particular regarding students' misconceptions of programming concepts [20, 26].

Visual programming languages, and especially block-based ones, are often used to alleviate the first two aforementioned difficulties. These programming languages are designed to allow learners to focus on the structure of an algorithm rather than on the syntax required to implement such algorithm. A comparative study between block-based and text-based programming languages showed that the use of a block-based programming language led to a better understanding of key programming notions [28]. This result explains the many studies benefiting from the use of block-based programming languages in introductory programming courses [17, 29].

Constructionism [18] is a theory often cited in Computer Science Education research which stipulates that "building knowledge structures [...] happen especially felicitously in a context where the learner is consciously engaged in constructing a public entity" [18, p. 1]. This theory is sometimes used to tackle the third difficulty we previously mentioned: there is a strong relation between designing a programmable object and seeing its behavior responds to changes in its program. As a result, constructionism paved the way to numerous studies using programmable objects in introductory programming courses. In the literature, such programmable objects could take two different shapes: they could either be tangible (e.g., LEGO Mindstorms [3], .NET Gadgeteer [9], or Thymio II [14]) or digital (e.g., Scratch [21] or Alice [4]).

Since physical computing concerns the design of tangible and programmable object, it is in a way inspired by constructionism. The literature review on physical computing highlights several advantages: increased learners' motivation and creativity [24], the inclusion of underrepresented minorities in Computer Science [23], and learning gains [3]. However, similar results can be also found when digital programmable objects are used to introduce Computer Science and programming [4, 15, 17].

Both situations (physical and digital computing) lead to learning gains, yet there is (to the best of our knowledge) no study trying to assess whether a situation is more beneficial than the other. The literature on Computer Science Education presents few studies using both a tangible programmable object and an equivalent simulation [2, 6, 8]. However, these articles do not compare learning gains but rather describe a system in which tangible and digital objects are complementary.

In this study, we wish to fill this gap in the literature by comparing the learning gains obtained from programming a tangible object or an identical simulation. Therefore, we ask the following research question: "are there differences in learning gains when programming a tangible object or an equivalent simulation of it?" In particular, we focused on the learning of three essential programming notions namely variables, conditional structures, and iterative structures. We thus re-formulate the previous question to focus on the learning of these three concepts.

To answer these three questions, we have developed a programming environment (described in the following section) with the objective of providing an identical user experience whether the user programs a tangible object or its simulation.

3 PROGRAMMING ENVIRONMENT

The programming environment used in our experiment consists of two components: the BBC micro:bit as the programmable object and the programming Web interface which implements a block-based programming language. Because we wish to compare the effects of physical and digital computing on students' learning gains, this programming environment comes in two versions: one for programming the tangible BBC micro:bit and the other for programming its simulation. These two versions provide an almost identical programming user experience as described in the following sections.

3.1 Programmable Object

The BBC micro:bit[1] is a programmable electronic board first released in 2016 in the United Kingdom. The objective of such an initiative was to make programming more accessible to teenagers by distributing these devices to middle school pupils aged 12-13 for free [1]. Since then, the BBC micro:bit has been used in several research projects related to introductory Computer Science courses, with positive results regarding the students' motivation, creativity, and basic knowledge of programming [22, 24, 25].

The micro:bit is a 4×5 cm pocket device that is equipped with a 256 kB flash memory and an ARM processor that controls all the components on the board. Some of these components are used to manage inputs and outputs. In particular, the micro:bit has an accelerometer, two push buttons, and a 5×5 grid of LEDs (see Figure 1). Because we target elementary programming notions, components that require at least a basic understanding of event-driven computing were not used in our experiment. As a consequence, we mainly took advantage of the LEDs to display information to the users. These LEDs can display alphanumeric characters or predefined shapes (see, for example, the heart in Figure 2, Zone 3). Longer texts can also be displayed on this grid thanks to a scrolling mechanism.

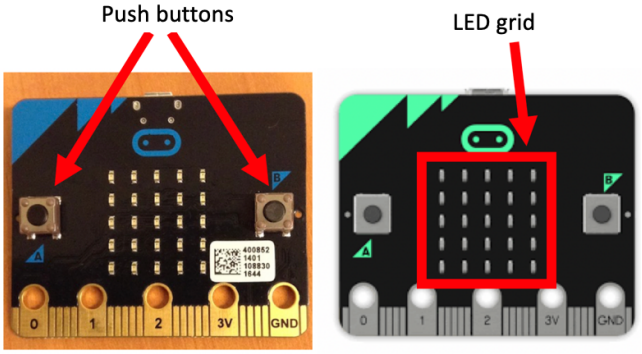

Figure 1: On the left, the tangible micro:bit card. On the right, a digital simulation.

To simulate the micro:bit, open source programs can be found on the Internet. For our experiment, we have exploited a program written in Javascript[2] which allowed us to easily implement the micro:bit simulation in our programming Web interface. This simulation was mainly used to display messages with the LEDs but it could also have replicated the effects of pressing the push buttons or activating the accelerometer if we wanted to use these components.

[1] https://microbit.org/
[2] https://github.com/Microsoft/pxt-microbit

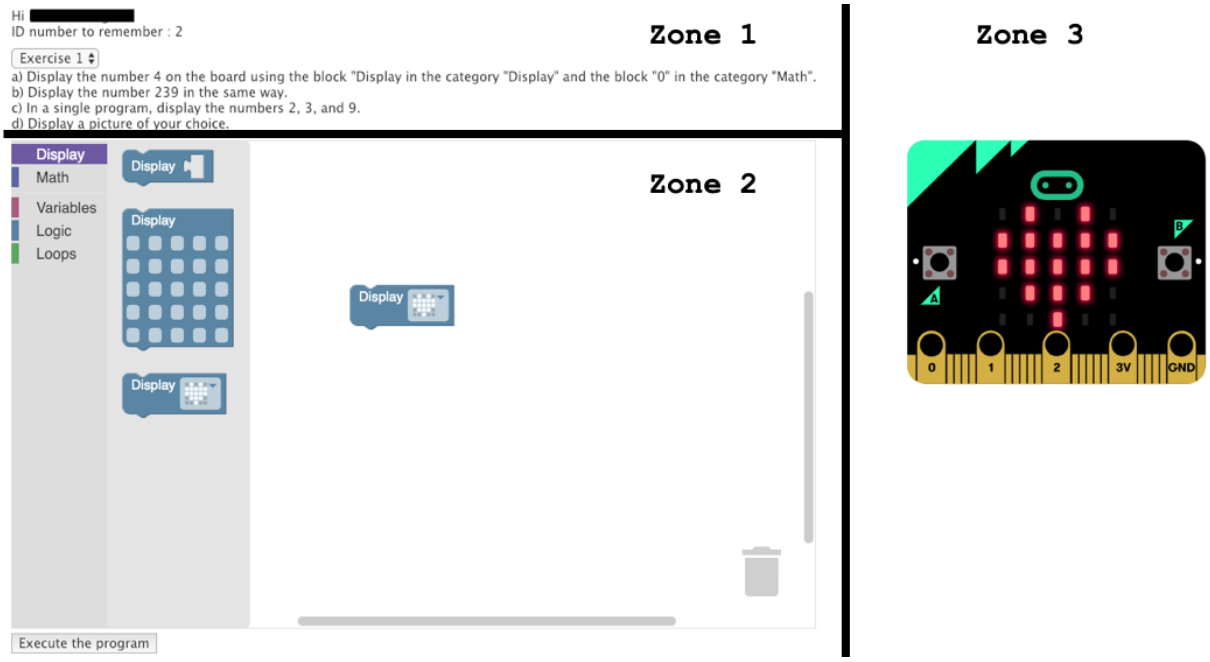

Figure 2: The block-based Web interface for programming the simulation of the micro:bit.

The execution of a program on the tangible micro:bit requires that it is connected to a computer via a USB cable. Its firmware can be updated by transferring a binary file to the flash memory of the micro:bit; this file actually contains the new program to be executed. The process of executing a program on the simulation is a little different: Javascript functions we have implemented are called to update the behavior of the simulation displayed on the screen. The internal process of executing a program on the tangible or simulated micro:bits might be different but, as explained in Section 3.2, it is invisible to the user: the simple action of pressing a button on the programming Web interface will let the user execute a program in both situations (with a tangible or a simulated micro:bit).

The two versions of the micro:bit can generate identical results for a same program. This was done to ensure an identical programming experience for the user regardless of the programmable object used. The only difference between the tangible and digital micro:bit lies in the space in which they can be manipulated. The interactions with the physical micro:bit can happen in a three-dimensional space: the board can be moved or flipped over. However, the digital micro:bit is statically represented in two dimensions on the computer screen: it cannot be moved nor rotated around any axis.

3.2 Programming Interface

We have developed a programming Web interface to program the micro:bit (see Figure 2). This interface implements Blockly[3], a library that allows users to write code using a block-based visual programming language. Two versions of this interface were also developed: one for programming the tangible micro:bit and one for programming its simulation. However these two interfaces differ in only a small part as described later in this section. Figure 2 illustrates the interface designed to program the micro:bit simulation. It can be divided in 3 main zones of interest.

Zone 1 displays some identification information related to the current user and a drop down menu which can be used to select a programming exercise out of all the proposed ones. The wording of the currently selected exercise is displayed beneath this drop down menu. These exercises are the same whether the user is programming the tangible or simulated micro:bit. The design of these exercises will be detailed in Section 4.2.

Zone 2 corresponds to the code edition area. The left-hand part in grey displays all the categories of blocks that can be used with this programming interface. For this experiment, we proposed five different categories: Display, Math, Variables, Logic, and Loops. Each category contains blocks that are specific to their theme. As an example, we will explain in details the "Display" category showed in Figure 2. This category contains three different blocks:

- the first one must be completed with another block in order to display a number on the micro:bit;
- the second block of this category allows the user to design a shape and to display it on the board;
- the last block can be used to display a pre-configured pictograph.

A user can build a program by dragging and dropping blocks from whichever category into the workspace (with the white background). A user can also create more complex programs by combining several different blocks like shown in Figure 3.

Zone 3 is the only zone of the programming interface that is different between the tangible and the simulation case. If the user programs the simulated micro:bit, Zone 3 displays the simulation

[3]https://developers.google.com/blockly/

which is updated each time the program is executed. If the user programs the tangible micro:bit, the simulation is replaced with instructions on how to execute a program on the tangible device.

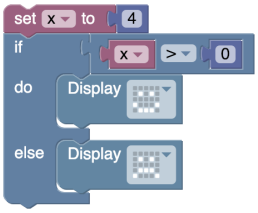

Figure 3: An example of a complex program written with multiple blocks.

We have made the process of executing the program on the micro:bit identical whether the user is programming the tangible or simulated device. Indeed, the user simply needs to click on the button located at the bottom of Zone 2. If programming the simulation, the program is directly executed on the display. If programming the tangible micro:bit, this leads to the block-based program being converted into a binary file and automatically transferred to the device. This results in an identical user experience whether the programmable object used is the tangible or simulated one.

4 EXPERIMENTAL PROTOCOL

To provide answers to our research questions, we have conducted an experimentation with teenagers aged 14-17 ($n = 36$, $\mu = 15.6$ year old, with 9 girls and 27 boys). We asked the participants to assess their prior knowledge of programming before the experiment took place. Of the 36 participants, 23 declared having "no prior knoledge" and 13 affirmed having a "basic understanding" of programming. These participants were divided into two groups: one programming the tangible micro:bit and the other programming its simulation. The group composition has been defined prior to the experiment so as to balance both groups in terms of age, gender, and prior knowledge. This distribution was validated by a post-experiment analysis, which is presented in Section 5.1. The experimental protocol designed for this study is composed of a pre- and post-test questionnaire and a set of programming exercises (as seen in Figure 2, Zone 1). The questionnaire and exercises targeted the notions of variables, conditional structures, and iterative structures. Since we did not make any assumption, the hypothesis we wanted to test was that programming a tangible object or its simulation causes different learning gains for each of these concepts.

The experiment went through four phases. The first phase consisted in a presentation to our participants of the programming Web interface and its block-based programming language. During this phase, we did not introduce any concept of programming except for the display of information on the grid of LEDs. Indeed, this was necessary for the participants to understand the pre-test questionnaire, which was administered during the second phase of the experiment. The third phase consisted in working on the programming exercises. Finally, the fourth phase concerned the answering of the post-test questionnaire. During the whole duration of the experiment, participants were asked to work individually and we would intervene only to provide a technical support.

4.1 Pre-Test and Post-Test Questionnaire

The pre- and post-test questionnaire was composed of 14 multiple choice questions: five on the concept of variables, five on conditional structures, and four on iterative structures. For each notion, the questions were ordered in an increasing difficulty. We based this ordering and the design of the questionnaire on the revised Bloom's taxonomy [11].

Each question asked for the result of a small program written with blocks. The blocks used in this questionnaire actually were identical to the ones implemented in the programming Web interface. This design decision was made to suppress the risk of misunderstanding the programs written in each question.

These questions were also designed to require knowledge of a single and unique programming notion at a time. For example, we did not introduce questions using the notion of loop variable because it would necessitate to use knowledge related to both variables and iterative structures. This detail is important in the design of our study because it would have been much more difficult to identify separately the learning gains for each concept otherwise. Figure 4 illustrates three questions that we used, one for each concept (ie., variables, conditionals, and loops).

Each question only has a single correct answer. We based the design of the other wrong answers on the literature review on students' misconceptions [20, 26]. This makes for plausible answers that participants might be inclined to select.

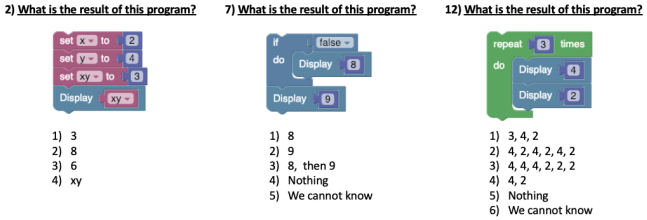

Figure 4: Three examples of questions used in the pre- and post-test questionnaire: left for variables, center for conditionals, right for loops.

4.2 Programming Exercises

During the third phase of the experiment, participants had 15 programming exercises to solve. Of these 15 exercises, four were on variables, three on conditional structures, and four on iterative structures. We also designed an exercise which was to be solved before all the others to familiarize our participants to the programming environment. We proposed another three exercises mixing concepts which were to be solved after all the others.

The design of the programming exercises used the same approach we have described in Section 4.1 for the pre- and post-test questionnaire. We used Qian and Lehman's literature review [20] to design exercises that could lead our participants into making mistakes and learning from them. The exercises were also proposed in an increasing level of difficulty; for each concept, the level of details provided in the wordings decreased while the number of blocks to manipulate increased from an exercise to the following

one. Figure 5 illustrates this last point with two exercises of different levels of difficulties: Exercise 5 corresponds to the last exercise on variables and Exercise 6 is the first one on conditional structures. Exercise 6 explicitly tells the learner which blocks should be used to solve the exercise while the wording of Exercise 5 is less detailed.

Participants were asked to do the exercises in the order they were proposed. However, our interface did not implement any automated assessment tool and the participants were free to go to the next exercise as they pleased. They were also authorized to go back to any previous exercise to see or modify their solutions as they were displayed back in the editor in Zone 2 of the programming interface.

This last point implies that the states of the exercises are saved from an exercise to the other. We also have implemented functions to save all the participants' interactions with the programming interface. These logs are not detailed in this article as they will be used in a later stage of this research project.

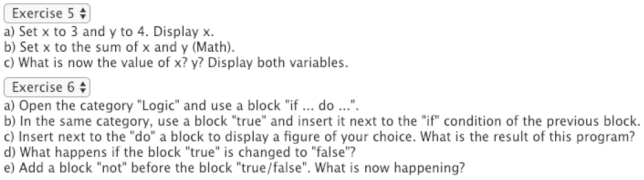

Figure 5: Two examples of questions with different levels of details in their wordings.

5 RESULTS

5.1 Validation of the Experimental Protocol

As mentioned in Section 4, the experiment gathered 36 participants aged 14-17. These participants were separated into two groups:

- The first group programmed the tangible micro:bit ($n = 18$, $\mu = 15.6$ year old). This group was composed of five girls and 13 boys. Eleven participants declared having no prior knowledge of programming and seven having an elementary understanding of programming.
- The second group programmed the simulation of the micro:bit ($n = 18$, $\mu = 14.8$ year old). This group was composed of four girls and 14 boys. In this group, 12 participants declared having no prior knowledge of programming and six having an elementary understanding of programming.

Because the participants were divided into groups before the start of the experiment, we wanted to verify that this distribution did not introduce a preliminary bias regarding the average initial programming knowledge of each group. Thus, we performed a two-tailed unpaired t-test to verify that the two groups were homogeneous in terms of initial programming knowledge. The results of this test ($t = 1.499, p = 0.145 > 0.05$) indicate that the mean scores obtained by each group for the pre-test were not significantly different. This first result confirms that the *a priori* group compositions did not introduce any initial bias in our experiment.

5.2 Learning Gains

Table 1 summarizes the scores obtained at the pre- and post-test.

5.2.1 Overall learning gains. We wanted to verify that participants to our experiment had significantly higher post-test scores, which would validate the design of our programming exercises. For this analysis, we performed two one-tailed paired t-tests: once for the group programming the tangible micro:bit and once for the group programming the simulated micro:bit. The results of the test ($t = 8.574, p = 7.014e\text{-}8 \ll 0.05$) suggest that the scores obtained at the post-test were significantly higher than those obtained at the pre-test for the participants programming the tangible micro:bit. A similar conclusion can be drawn for the participants programming the simulation of the micro:bit ($t = 5.242$, $p = 3.311e\text{-}5 \ll 0.05$). This aspect validates the design of our exercises to introduce elementary concepts to participants with little or no prior knowledge of programming.

We also wanted to compare the learning gains between each group by performing a two-tailed unpaired t-test. For each participant, the learning gain was calculated by subtracting the pre-test score to the post-test score. The results of this test ($t = 0.687$, $p = 0.497 > 0.05$) suggest that there was no significant difference between the average learning gains of each group.

5.2.2 Variables. We first evaluated the learning gains related to the concept of variables. We have used, for both groups, a one-tailed paired t-test to determine whether the results of the post-test were better than those of the pre-test. Regarding the group programming the tangible object, no significant increase could be found ($t = 0.325$, $p = 0.375 > 0.05$). The results concerning the group programming the simulation are similar ($t = 1.046, p = 0.155 > 0.05$).

Since both groups did not significantly improve their scores from pre-test to post-test, we did not pursue our analysis any further with a comparison of the learning gains between each group.

5.2.3 Conditional Structures. We have performed a similar analysis regarding the concept of conditional structures. For the group programming the tangible object, the results of the one-tailed paired t-test ($t = 6.556, p = 2.449e\text{-}6 \ll 0.05$) show that the scores have significantly improved from pre-test to post-test. A similar conclusion can be drawn for the group programming the simulation ($t = 6.761$, $p = 1.664e\text{-}6 \ll 0.05$). These results suggest that both groups have improved their understandings of conditional structures.

We have continued our analysis to highlight whether one group had higher learning gains than the other. Since we do not make any assumption as to which programming environment yields better results, we have performed a two-tailed unpaired t-test. The results ($t = 1.425, p = 0.163 > 0.05$) suggest that there is no significant difference in learning gains on conditional structures when a student programs a tangible object or a simulation.

5.2.4 Iterative Structures. Finally, we have run the same tests for the concept of iterative structures. The results of the one-tailed paired t-tests ($t = 6.174$ and $p = 5.099e\text{-}6 \ll 0.05$ for the group programming the tangible micro:bit, and $t = 3.915$ and $p = 5.579e\text{-}4 \ll 0.05$ for the group programming the simulated micro:bit) show that both groups significantly improved their scores from pre-test to post-test, indicating a better understanding of iterative structures.

However, and once again, the results of the two-tailed unpaired t-test ($t = 0.586, p = 0.562 > 0.05$) show that the two programming

Table 1: Results of the pre- and post-test for each group of participants.

Group	Variables (5 pts)		Conditional Structures (5 pts)		Iterative Structures (4 pts)	
	Pre-test	Post-test	Pre-test	Post-test	Pre-test	Post-test
Tangible micro:bit	$\mu = 4.167$	$\mu = 4.222$	$\mu = 1.667$	$\mu = 4.167$	$\mu = 2.389$	$\mu = 3.444$
	$\sigma^2 = 0.735$	$\sigma^2 = 0.889$	$\sigma^2 = 1.059$	$\sigma^2 = 1.441$	$\sigma^2 = 0.840$	$\sigma^2 = 0.497$
Simulated micro:bit	$\mu = 3.611$	$\mu = 4.000$	$\mu = 1.389$	$\mu = 3.222$	$\mu = 2.222$	$\mu = 3.111$
	$\sigma^2 = 2.605$	$\sigma^2 = 0.941$	$\sigma^2 = 0.605$	$\sigma^2 = 1.595$	$\sigma^2 = 0.771$	$\sigma^2 = 0.928$

environments seem to lead to equivalent learning gains regarding iterative structures.

6 DISCUSSION

The results of our study suggest that, while both groups significantly improved their scores at the post-test, there is no significant difference in learning gains when a student programs a tangible object or an exact equivalent simulation of it. A possible explanation for these results could be the fact that the experiment did not heavily rely on the manipulation of the tangible micro:bit. Indeed, the design of our exercises did not encourage participants to interact physically with the tangible device and the handling of the micro:bit had no impact on the program being executed. Still, these results question the benefits of using physical computing to learn elementary programming concepts as mentioned in the literature.

Our study involves participants that are older than the students usually initiated to programming with physical computing. The rationale behind this design decision is that we wanted to have participants that could progress in the activities without our assistance. Indeed, we did not teach any preliminary knowledge nor helped the participants during the experiment. Moreover, this let us have a really controlled set-up for this initial study about comparing learning gains using a tangible object or its simulation.

Regarding the concept of variable, results did not show any significant learning gain between the pre-test and the post-test. Indeed, both groups obtained good pre-test scores as shown in Table 1, which left little space for improvement in the post-test. We identified two possible reasons: the questions were too simple or the notion of variable is much easier to understand than the others. This last point can actually also be found in other studies [29].

Our experiment presents several possible points of improvement. Firstly, participants could interact with the programming environment only for half a day. Although our results show a significant learning gain for each group, it would be interesting to analyze data from a longer experiment conducted for example in an institutional context. Such an experiment would make it possible to comment on the interest of using programmable tangible objects in introductory programming courses.

Secondly, while our experiment involved a non-negligible sample size, it could be interesting to run a study with more participants. This would allow us to have a more representative sample from a statistical point of view. The results of such an experiment would complement those presented in this article and would also provide a stronger conclusion regarding the interest of using physical computing for the introduction and learning of programming.

7 CONCLUSION AND FUTURE WORK

In this article, we presented a study which compared the learning gains when a learner programs a tangible object or its simulation. The results indicate that, although a significant learning is observed in both cases, these situations appear to be broadly equivalent in terms of overall programming learning gains. More specifically, we could not observe a significant difference between these two situations regarding the learning of conditional and iterative structures. This conclusion questions us about the intrinsic benefits of physical programming sometimes mentioned in the literature. To further explore the potential benefits of physical and digital programming we plan to, in the future, improve our experimental set-up as follow.

The first area for future work concerns the study of learners' behaviors when they face a programming environment involving a tangible object or a simulation. During our experiment, learners had to answer a meta-cognitive [7, 13] questionnaire designed to evaluate how they situate themselves in the learning process [30]. In particular, we wish to evaluate if they experienced a flow [10] while learning programming and if this kind of exercise could improve their empowerment [12]. In this way, we hope to be able to provide additional answers about the extent of motivation and creativity when learners use physical and/or digital programming.

During the experiment, we also recorded the users interactions with the programming interfaces: all the participants' actions and mouse positions were logged. We are going to analyze this data to search for patterns or strategies used by learners to solve the programming exercises. Moreover, by looking at the participants' actions from one exercise to the other, we also aim to highlight the evolution of their understandings of a specific programming concept. Finally, this analysis will be coupled with the results of the pre- and post-tests to find possible correlations between learning strategies and learning gains.

Finally, we wish to conduct two new experiments. The first one will improve the study presented in this article by addressing the lack of physical interactions previously discussed. With a similar set-up (i.e., homogeneity of initial knowledge, same age group, balanced distribution between girls and boys), this new experiment will benefit from the push buttons and accelerometer present on the micro:bit and mentioned in Section 3.1 to engage participants in physically interacting with the tangible board. The second experiment will be designed to identify learning gains differences when physical computing is used in STEM (Science, Technology, Engineering, Mathematics). Some articles already present this type of practice [16, 27] and the idea is to conduct a similar comparative study with younger participants to highlight the benefits of physical computing in learning STEM notions.

REFERENCES

[1] Thomas Ball, Jonathan Protzenko, Judith Bishop, Michal Moskal, Jonathan de Halleux, Michael Braun, Steve Hodges, and Claire Riley. 2016. Microsoft Touch Develop and the BBC micro:bit. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, Austin, TX, USA, 637–640.

[2] Philipp Brauner, Thiemo Leonhardt, Martina Ziefle, and Ulrik Schroeder. 2010. The Effect of Tangible Artifacts, Gender and Subjective Technical Competence on Teaching Programming to Seventh Graders. In *Teaching Fundamentals Concepts of Informatics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 61–71.

[3] Daniel C. Cliburn. 2006. Experiences with the LEGO Mindstorms Throughout the Undergraduate Computer Science Curriculum. In *Proceedings. Frontiers in Education. 36th Annual Conference*. IEEE, San Diego, CA, USA, 1–6.

[4] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D Tool for Introductory Programming Concepts. In *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges (CCSC '00)*. Consortium for Computing Sciences in Colleges, USA, 107–116. http://dl.acm.org/citation.cfm?id=364132.364161

[5] Benedict du Boulay. 1986. Some Difficulties of Learning to Program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73.

[6] Barry Fagin. 2003. Ada/Mindstorms 3.0. *IEEE Robotics & Automation Magazine* 10, 2 (2003), 19–24.

[7] John H. Flavell. 1979. Metacognition and cognitive monitoring: a new area of cognitive-developmental inquiry. *American Psychologist* 34, 10 (1979), 906–911.

[8] Aaron Garrett and David Thornton. 2005. A Web-Based Programming Environment for LEGO Mindstorms Robots. In *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 2 (ACM-SE 43)*. ACM, New York, NY, USA, 349–350. https://doi.org/10.1145/1167253.1167333

[9] Steve Hodges, James Scott, Sue Sentance, Colin Miller, Nicolas Villar, Scarlet Schwiderski-Grosche, Kerry Hammil, and Steven Johnston. 2013. .NET Gadgeteer: A New Platform for K-12 Computer Science Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 391–396. https://doi.org/10.1145/2445196.2445315

[10] Paolo Inghilleri, Giuseppe Riva, and Eleonora Riva. 2018. *Enabling Positive Change*. https://content.sciendo.com/view/title/506140

[11] David R. Krathwohl. 2002. A Revision of Bloom's Taxonomy: An Overview. *Theory Into Practice* 41, 4 (2002), 212–218.

[12] Creative Commons Attribution ShareAlike 4.0 International License. 2015. *Introduction to Physical Computing*. Technical Report. Maker Education Initiative.

[13] Jennifer Livingston. 2003. Metacognition: An Overview. (01 2003).

[14] Stéphane Magnenat, Fanny Riedo, Michael Bonani, and Francesco Mondada. 2012. A Programming Workshop Using the Robot "Thymio II": The Effect on the Understanding by Children. In *2012 IEEE Workshop on Advanced Robotics and its Social Impacts*. IEEE, Munich, Germany, 24–29.

[15] Louis Major, Theocharis Kyriacou, and Pearl Brereton. 2014. The Effectiveness of Simulated Robots for Supporting the Learning of Introductory Programming: a Multi-Case Case Study. *Computer Science Education* 24, 2-3 (2014), 193–228.

[16] Michela Maschietto and Sophie Soury-Lavergne. 2013. Designing a Duo of Material and Digital Artifacts: the Pascaline and Cabri Elem e-Books in Primary School Mathematics. *ZDM* 45, 7 (2013), 959–971.

[17] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2013. Learning Computer Science Concepts with Scratch. *Computer Science Education* 23, 3 (2013), 239–264.

[18] Seymour Papert and Idit Harel. 1991. Situating Constructionism. *Constructionism* 36, 2 (1991), 1–11.

[19] Mareen Przybylla and Ralf Romeike. 2014. Key Competences with Physical Computing. *KEYCIT 2014–Key Competencies in Informatics and ICT (Preliminary Proceedings)* (2014), 216.

[20] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (Oct. 2017), 24 pages. https://doi.org/10.1145/3077618

[21] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (2009), 60–67.

[22] Albrecht Schmidt. 2016. Increasing Computer Literacy with the BBC micro:bit. *IEEE Pervasive Computing* 15, 2 (2016), 5–7.

[23] Sue Sentance and Scarlet Schwiderski-Grosche. 2012. Challenge and Creativity: Using .NET Gadgeteer in Schools. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education (WiPSCE '12)*. ACM, New York, NY, USA, 90–100. https://doi.org/10.1145/2481449.2481473

[24] Sue Sentance, Jane Waite, Steve Hodges, Emily MacLeod, and Lucy Yeomans. 2017. Creating Cool Stuff: Pupils' Experience of the BBC micro:bit. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE '17)*. ACM, New York, NY, USA, 531–536. https://doi.org/10.1145/3017680.3017749

[25] Sue Sentance, Jane Waite, Lucy Yeomans, and Emily MacLeod. 2017. Teaching with Physical Computing Devices: the BBC micro:bit Initiative. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education (WiPSCE '17)*. ACM, New York, NY, USA, 87–96. https://doi.org/10.1145/3137065.3137083

[26] Juha Sorva. 2012. *Visual program simulation in introductory programming education*. Aalto University.

[27] Patrick Wang, Ilaria Renna, Frédéric Amiel, and Xun Zhang. 2018. Learning with Robots in CS and STEM Education: A Case Study with ISEP-R0B0. In *Proceedings of the 4th Workshop on Robots for Learning at ACM/IEEE HRI 2018*. 16–21.

[28] David Weintrop and Uri Wilensky. 2015. Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research (ICER '15)*. ACM, New York, NY, USA, 101–110. https://doi.org/10.1145/2787622.2787721

[29] David Weintrop and Uri Wilensky. 2017. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. Comput. Educ.* 18, 1, Article 3 (Oct. 2017), 25 pages. https://doi.org/10.1145/3089799

[30] Philip H. Winne. 2005. A perspective on state-of-the-art research on self-regulated learning. *Instructional Science* 33, 5–6 (2005), 559–565.