Computer Science Education
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ncse20

The role of physicality in rich programming environments

Allison S. Liu[a], Christian D. Schunn[a], Jesse Flot[b] & Robin Shoop[b]
[a] Department of Psychology, Learning Research and Development Center, University of Pittsburgh, Pittsburgh, PA, USA
[b] The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA

Published online: 21 Oct 2013.

Computer Science Education, 2013
http://dx.doi.org/10.1080/08993408.2013.847165

The role of physicality in rich programming environments

Allison S. Liu[a]*, Christian D. Schunn[a], Jesse Flot[b] and Robin Shoop[b]

[a]*Department of Psychology, Learning Research and Development Center, University of Pittsburgh, Pittsburgh, PA, USA;* [b]*The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA*

(*Received 28 May 2013; accepted 30 July 2013*)

Computer science proficiency continues to grow in importance, while the number of students entering computer science-related fields declines. Many rich programming environments have been created to motivate student interest and expertise in computer science. In the current study, we investigated whether a recently created environment, Robot Virtual Worlds (RVWs), can be used to teach computer science principles within a robotics context by examining its use in high-school classrooms. We also investigated whether the lack of physicality in these environments impacts student learning by comparing classrooms that used either virtual or physical robots for the RVW curriculum. Results suggest that the RVW environment leads to significant gains in computer science knowledge, that virtual robots lead to faster learning, and that physical robots may have some influence on algorithmic thinking. We discuss the implications of physicality in these programming environments for learning computer science.

Keywords: algorithmic thinking; computer programming; physicality; programming environments; robotics; virtual simulations

Introduction

Computer science is a key field for advancement and innovation in science, technology, engineering, and mathematics fields, and knowledge of computer science and programming is increasingly important for literacy in today's society. However, as computer science grows in importance, a declining number of students are entering computer science-related fields (Dann, Cooper, & Pausch, 2006; Vesgo, 2005), Furthermore, among computer science students, average levels of performance are much lower than expected. For example, when McCracken and colleagues (2001) examined the performance of first-year computer science students after students had completed their introductory courses, they found that students

*Corresponding author. Email: asl36@pitt.edu achieved an average score of 20.8% on the study's assessment. In another investigation of novice programmers' code reading skills, Lister and colleagues (2004) found similarly low performance. The authors concluded that many computer science students do not possess the fundamental knowledge required for problem solving in programming.

Rich programming environments

The need to motivate and improve student involvement and learning in computer science has led to the creation of many rich programming environments. These visual environments, generally targeted at younger audiences, are intended to be highly interactive and to immerse students while teaching programming and other skills, such as analytic thinking and creativity. Many programming environments currently exist, including numerous implementations of the Logo programming language (Papert, 1980), Scratch (Resnick et al., 2009), and Alice (Dann et al., 2006). A number of studies show that these programming environments can be used to successfully teach and improve students' programming and computer science abilities (e.g. Bishop-Clark, Courte, & Howard, 2006; Clements, 1990; Meerbaum-Salant, Armoni, & Ben-Ari, 2010; Sutherland, 1993; Sykes, 2007).

Recently, Carnegie Mellon University and Robomatter Inc. have developed a new programming environment situated in the context of robotics, named Robot Virtual Worlds (RVWs). The RVW environment allows students to program virtual robots (including LEGO, VEX, TETRIX and fantasy robots) using ROBOTC, a C-based robot programming language. Students can then use their robots in simulated three-dimensional worlds, which consist of themed worlds (e.g. a tropical island, an underwater world, a different planet) and table-top boards; one themed and one table-top world are shown in Figure 1. These worlds include built-in tasks of varying difficultly that students can complete. Users are also able to download additional tools that work with the RVW environment, such as a level designer for students to create their own stages, object importers for students to build and import their own three-dimensional objects into their RVW levels, and a measurement tool kit to aid path planning and navigation. The RVW environment is comparable with a physical robotics environment in many ways. RVW runs off of Unity, which includes a built-in physics engine that can simulate a physical robot's wheel slippage and effector error. Although RVW does not include artificial sensor noise, it can still simulate the range of sensor values that students would normally encounter with a physical robot (e.g. a value of −1 when the sensor is out of range), creating scenarios that require advanced thinking from the programmer.

The RVW environment is also complemented by an online curriculum that leads students through robotics programming. The course is made up of nine units. Each unit consists of video lectures, paper-based activities, online

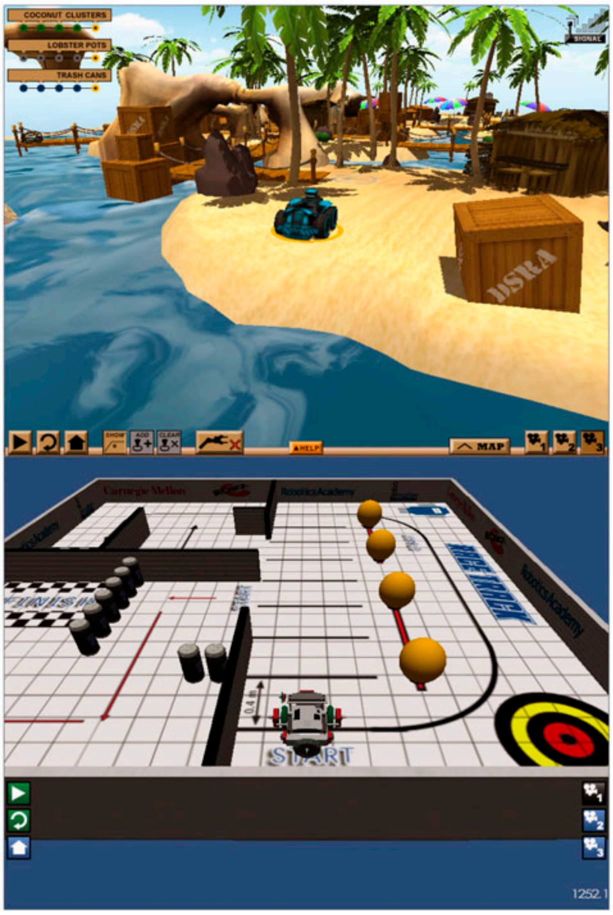

Figure 1. Two of the three-dimensional worlds in which students can program their robots in the RVW environment.

practice quizzes and table-top challenges that can be completed using either the RVW simulations or with physical materials. Unit topics include: how to use the RVW software, how the physical robot functions, robot programming basics (e.g. how to program the robot to move forward at different speeds), and more advanced programming concepts (e.g. using functions and sensors). Because robotics has been shown to be an effective tool for teaching introductory computer science concepts (see Major, Kyriacou, & Brereton, 2011 for a review), we are interested in whether this new environment can improve students' computer science and robotics programming knowledge.

Affordances of virtual environments

Virtually simulated environments, such as the RVW environment, may provide several unique learning affordances that physical environments cannot. For example, virtual environments allow students to quickly see their program in action. Students can also pause their program simulations as soon as they encounter an error, allowing them to remedy the error without delay. Meanwhile, in a physical environment, students may not be able to stop their programs as quickly or as easily, leading to more delayed feedback and delayed error correction. Teaching robotics programming in classrooms is a particular challenge, as physical robotics materials can be difficult to obtain due to cost and the amount of space required to store them. Physical robots also require additional time to build and set up before they can be used (Major et al., 2011), and students are generally unable to access these robots outside of class time, further limiting the amount of time students are able to spend learning with the robot. Thus, virtual robotics simulations may be especially beneficial for their accessibility and convenience.

However, physical environments have also been shown to have unique learning affordances. For instance, physical environments provide a perceptual space to ground abstract concepts, and they provide experience with errors caused by physical artefacts (see Olympiou & Zacharia, 2012). In the robotics context, the physical environment may allow students to more clearly see the connection between their program code and the actions of the robot itself. A physical robot may also provide more motivation for students whose interests lie in robotics, as working with a virtual simulation may feel less authentic than working with an actual robot. Using the RVW environment's ability to utilize the same ROBOTC code for both virtual and physical robots, we investigated whether physicality (or lack thereof) in these rich virtual environments has any influence on learning computer science concepts.

Research objectives

The current paper asks two questions: whether the RVW environment can be used to improve students' knowledge of computer science concepts; and whether physicality plays a role in students' learning through the RVW environment. In order to answer these questions, we conducted two studies. In Study 1, we investigated the use of the RVW environment and online curriculum for teaching computer science and robotics programming in several high-school classrooms. In Study 2, we investigated whether students who use virtual robots vs. physical robots to complete the online curriculum differed in their learning.

Study 1

In Study 1, we attempted to verify that the RVW environment and complementary online curriculum could allow students to learn robotics programming and computer science principles.

Methods

Participants

Three public high school classes completed the RVW online curriculum using VEX robots. The three courses were elective robotics courses from three different schools. The courses, hereby designated as "Class V + P," "Class V1" and "Class V2" consisted of 10, 23 and 13 students, respectively, for a total of 46 students. The majority of students were in their freshman or sophomore year, with little to no prior programming experience. Class V + P completed the curriculum's exercises through the RVW software and then repeated the exercises using a physical VEX robot. Classes V1 and V2 used only the virtual VEX robot simulation to complete the curriculum.

Data collection

Students completed a 50-question, multiple-choice pretest at the start of the course, and an identical post-test after completing the course (sample questions can be found in the Appendix). The test included four sub-categories of questions:

- General programming questions: Questions that involved syntax or concepts that are applicable to multiple programming languages. Example: "If the condition of an If statement is true, then all of the code inside of its curly braces will run. True/False."
- ROBOTC syntax questions: Questions that involved ROBOTC syntax or the ROBOTC application (e.g. using menus in the ROBOTC application). Example: "To make the robot stop, you set its motor values equal to __."
- Physical robot functioning questions: Questions that involved the physical VEX robot's functioning. Example: "The VEX Ultrasonic Rangefinder (sonar sensor) measures distance using __."
- Algorithmic thinking questions: Questions that involved thinking through the process of the programming problem (e.g. planning the program, using pseudocode, predicting the behaviour of a program) or more abstract concepts of programming. Example: "Given the program above, the robot will do ___."

The number of problems in each sub-category and the Cronbach's alpha (α, a commonly used metric of instrument reliability; Cronbach, 1951) for

Table 1. Number of items and Cronbach's alpha for question sub-categories.

Problem sub-category	Number of items	α
Algorithmic thinking	4	.54
General programming	13	.56
ROBOTC syntax	37	.84
Physical robot functioning	19	.81

each category are shown in Table 1. The sub-categories were not mutually exclusive; 23 of the 50 questions fell into two sub-categories. As a general rule of thumb, a minimum alpha coefficient of .7 or .8 is recommended (e.g. Nunnally, 1978). Note that the coefficients for algorithmic thinking and general programming fall below this suggested threshold. These low alpha coefficients could be indicative of an insufficient number of test items to test the latent construct, low interrelatedness between test items, or heterogeneous constructs underlying the test items, and must be considered as a potential limitation regarding the algorithmic thinking and general programming sub-categories.

Data analyses

Analyses were done separately for each class to examine consistency of results across contexts, and to control for class-level effects that would bias an overall learning gain analysis. For each class, we conducted a paired-samples *t*-test that compared students' pretest scores to their post-test scores, to see whether students significantly improved in their programming knowledge after completing the RVW course. We also conducted a separate *t*-test for each of the four sub-categories of questions, to examine whether learning differed across them. Due to the uneven number of problems in each category, we used the proportion of correct answers within each category as a measure of accuracy.

Results

Overall performance changes

Our first analysis investigated whether students improved in their computer science knowledge, using pretest and post-test as measures of performance. All three classes showed significant improvement from pretest to post-test [Class V + P: $t(9) = -9.5$, $p < .001$; Class V1: $t(22) = -14.4$, $p < .001$; Class V2: $t(12) = -7.1$, $p < .001$]. Figure 2 shows that the overall learning gain did not differ by pretest score, as almost all participants improved regardless of their pretest score. Mean gains were substantial: out of 100 possible points, Class V + P improved by an average of 24.8 points, Class

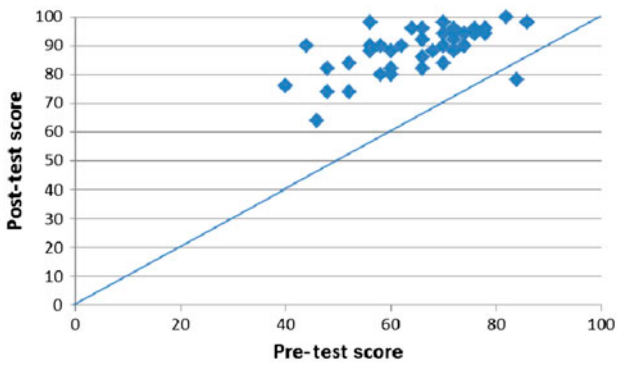

Figure 2. A comparison of Study 1 participants' total pretest scores plotted against their total post-test scores.

Table 2. Study 1 mean proportion correct (and standard deviations) for full test and question sub-categories, by class.

Class		Total	General programming	ROBOTC syntax	Physical robot	Algorithmic thinking
V + P	Pre	.62 (.15)	.65 (.24)	.60 (.16)	.61 (.19)	.80 (.20)
	Post	.87 (.10)***	.97 (.15)***	.87 (.10)***	.91 (.14)***	.98 (.08)*
V1	Pre	.65 (.11)	.65 (.11)	.63 (.11)	.69 (.14)	.89 (.15)
	Post	.90 (.07)***	.97 (.10)***	.89 (.07)***	.98 (.10)***	.96 (.12)
V2	Pre	.68 (.12)	.74 (.16)	.66 (.14)	.70 (.15)	.81 (.29)
	Post	.87 (.08)***	.93 (.12)***	.86 (.10)***	.95 (.07)***	.85 (.13)

Notes: Denotes significant increases from pre-test to post-test.
$^{*}p < .05$, $^{**}p < .01$, $^{***}p < .001$.

V1 improved by an average of 25.0 points, and Class V2 improved by an average of 19.1 points (see Table 2 for average pretest and post-test scores).

Changes within question sub-categories

To obtain a more nuanced look at students' improvements in different aspects of computer science and programming knowledge, we looked at students' pretest and post-test scores within each question sub-category (general programming, ROBOTC syntax, physical robot functioning and algorithmic thinking).

Students in all three classes showed significant improvements in general programming, ROBOTC syntax and physical robot functioning, from pre-scores of .75 or lower to post-scores of .85 or higher (see Table 2 for average pretest and post-test scores for each sub-category, and *p*-values). Only students in Class V + P showed a significant increase in algorithmic thinking [$t(9) = -2.3$, $p = .045$); in contrast, students from Class V1 only marginally significantly improved [$t(22) = -2.0$, $p = .056$], and students from Class V2 did not improve [$t(12) = -.56$, $p = .584$] (see Figure 3).

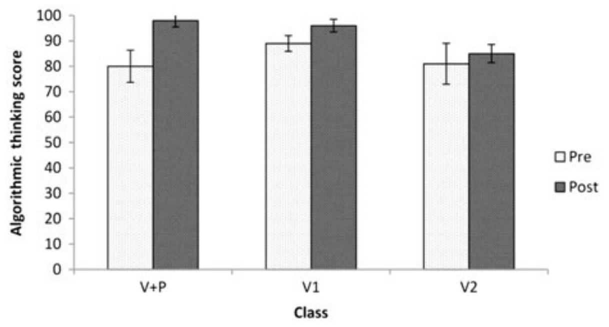

Figure 3. From Study 1, the average algorithmic thinking score on the pretest and post-test for Classes V + P, V1 and V2.

Discussion

Study 1 suggests that the RVW environment can be used to teach computer science and programming concepts. Students significantly improved their test performance after completing the online RVW curriculum. Thus, as with other rich programming environments, the RVW environment and its online curriculum are valid tools for teaching computer science concepts through robotics.

A closer examination of the question sub-categories showed that students were likely to improve in the topics of general programming, ROBOTC syntax and physical robot functioning. There were inconsistent gains in algorithmic thinking: only students in Class V + P significantly improved.

Interestingly, the two classes that showed no algorithmic thinking gains used only the virtual VEX robot provided by the RVW environment, while Class V + P used a combination of virtual and physical VEX robots. This result brings up the question as to whether the inclusion of the physical robot uniquely contributed to learning in algorithmic thinking, above and beyond the learning provided by the virtual VEX robot. As stated earlier, the physical robot may encourage a stronger association between students' ROBOTC code and the actions of their VEX robot. This association may allow them to better visualize and plan their programs, leading to greater algorithmic thinking gains as seen here. In Study 2, we further investigated the possible benefit provided by the physical VEX robot.

Study 2

In Study 2, we investigated the possibility that the physical VEX robot provides unique contributions to learning, specifically in algorithmic thinking. We contrasted two classes that completed the online RVW curriculum using only virtual VEX robots or only physical VEX robots.

Methods

Participants

Public high school students from two elective programming classes participated in the study, with the same instructor teaching both classes. One class of 15 students completed the course using virtual VEX robots and the virtually simulated table-top environments (the "Virtual" class). A second class of 11 students completed the RVW course using physical VEX robots and physical table-top boards that were visually matched with the virtual simulations (the "Physical" class). Both classes consisted primarily of freshmen and sophomores with little or no prior programming experience.

Data collection and analyses

Both classes completed the 50-item pretest and post-test used in Study 1, and both classes completed the pretest within one or two days of the same date. Students' total scores on the pretest were compared with their total scores on the post-test. To control for students' differing pretest scores, an analysis of covariance (ANCOVA) was run using condition (Virtual, Physical) as the independent variable, post-test score as the dependent variable, and pre-test score as the covariate. We also examined whether learning differed across the test's four question sub-categories (algorithmic thinking, general programming, ROBOTC syntax, physical robot) by performing a separate ANCOVA for each question sub-category.

In addition, we looked at the number of days between students' pretest attempt and post-test attempt. This was used as a measure of the time needed to complete the course, to see whether one condition required less time than the other to learn the same amount of information. A one-way analysis of variance (ANOVA) was performed, using condition (Virtual, Physical) as the independent variable and number of days as the dependent variable.

Results

Overall performance changes

The Virtual and Physical classes showed no differences in their overall post-test scores [$F(1, 23) = .19$, $p = .67$] when pretest scores were controlled.

Table 3. Study 2 mean proportion correct (and standard deviations) for full test and question sub-categories, by class.

Class		Total score	General programming	ROBOTC syntax	Physical robot	Algorithmic thinking
Virtual	Pre	.56 (.12)	.59 (.12)	.51 (.11)	.51 (.14)	.80 (.29)
	Post	.85 (.15)	.86 (.15)	.83 (.16)	.86 (.17)	.95 (.14)
Physical	Pre	.50 (.11)	.54 (.12)	.49 (.10)	.42 (.12)	.67 (.27)
	Post	.82 (.11)	.79 (.11)	.81 (.11)	.82 (.16)	.88 (.20)

Both classes began with similar pretest scores and ended with similar post-test scores. The average pretest and post-test scores for both classes, as well as their sub-category scores, can be found in Table 3.

Changes within question sub-categories

Unexpectedly, the Virtual and Physical classes did not show any learning differences across the four sub-categories of general programming [$F(1, 23) = 1.3$, $p = .27$], ROBOTC syntax [$F(1, 23) = .079$, $p = .78$], physical robots [$F(1, 23) = .11$, $p = .74$], or algorithmic thinking [$F(1, 23) = .061$, $p = .81$] when pretest scores were controlled. Follow-up paired-sample *t*-tests showed that the classes significantly improved in all four sub-categories after completing the online course (general programming: $t(25) = -7.6$, $p < .001$; ROBOTC syntax: $t(25) = -11.8$, $p < .001$; physical robot functioning: $t(25) = -11.9$, $p < .001$; algorithmic thinking: $t(25) = -4.4$, $p < .001$].

Time taken to complete the course

The Physical class took significantly more time than the Virtual class [$F(1, 24) = 30.3$, $p < .001$] to complete the RVW online curriculum. All students in the Physical class completed the course in the same amount of time: working with the Physical robots did not afford them the same freedom that students in the Virtual class had, who worked independently through the course with teacher support. Due to heterogeneity of variance, we ran an additional *t*-test with equal variances not assumed, which was significant [$t(24) = 6.5$, $p < .001$, $d = 2.2$]. The Physical class took an extra 30.3 days on average (approximately one month) to complete the online course than the Virtual class (see Figure 4).

To confirm that the time savings seen in the Virtual class were consistent, we also looked at two additional classes who completed the same

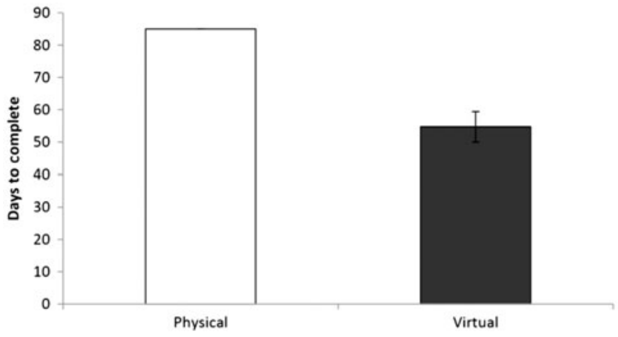

Figure 4. From Study 2, the number of days taken by the Physical class and the Virtual class to complete the RVW online curriculum.

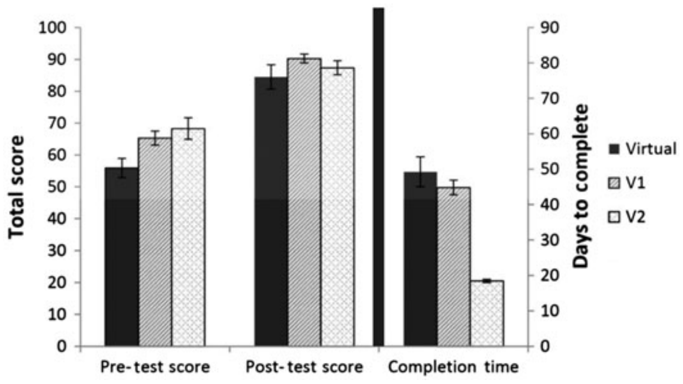

Figure 5. The total pretest scores, total post-test scores and number of days taken to complete the online RVW course for Study 2's Virtual class (Virtual) and the two Virtual-only classes from Study 1 (V1 and V2).

programming course with virtual VEX robots (Classes V1 and V2 from Study 1). Figure 5 shows a comparison of the three courses' pretest scores, post-test scores, and the number of days to complete the course.

To compare the time taken by each virtual class to complete the course, we ran a one-way ANOVA, using class (Class "Virtual" being the Virtual class in Study 2) as the independent variable, and number of days as the dependent variable. On average, Class Virtual took 54.7 days to complete the course (SD = 18.2), Class V1 took 49.8 days to complete the course (SD = 12.3), and Class V2 took 20.5 days to complete the course (SD = 1.7). The ANOVA was significant [$F(2, 48) = 31.6$, $p < .001$], and a *post hoc* contrast revealed that Class V2 took significantly less time than the other two classes to complete the online course. Due to heterogeneity of variance between Class V2 and the other two classes, we ran a second independent *t*-test comparing the amount of time taken by Classes Virtual and V1 (combined) and Class V2, which was significant [$t(49) = 13.2$, $p < .001$]. This suggests that all three Virtual classes allowed students to complete the course in significantly less time than the Physical class in Study 2.

Discussion

Both the Virtual class and the Physical class displayed equal learning gains, as their overall post-test scores were statistically equivalent (controlling for pretest scores). Unlike Study 1, there were no differences in algorithmic thinking gains between the Virtual and Physical classes, and both classes showed equal learning gains across the four sub-categories of questions. Possible reasons for the inconsistency between Study 1 and Study 2 are explored in the General Discussion.

The Virtual class showed a time reduction benefit, completing the course approximately one month earlier than the Physical class, with no effect on overall learning. This suggests that working with the virtual VEX robots allowed students to learn and complete the course more efficiently in the RVW environment when compared to the physical VEX robots.

The course instructor's informal observations support the efficiency of the virtual robots. He noted that students in the Physical class encountered additional communication, electrical, and mechanical issues with the physical robots, requiring the instructor to devote his time to troubleshooting problems. Part of the class's time was also taken up by setup and clean up, further impeding the time spent on programming. Because the virtual robots did not have the same physical limitations, the instructor and his students could focus their attention solely on programming. Students were also able to work from home to spend additional time learning outside of class.

General discussion

The current studies investigated two primary questions: whether the RVW environment and its online curriculum could be used to effectively teach computer science and robotics programming principles, and whether the lack of physicality in the RVW environment affected student learning. Study 1 provides evidence that the use of the RVW environment and its curriculum can lead to substantial learning gains, especially in general programming concepts, ROBOTC's syntax and the workings of the VEX robot. This is consistent with studies on other rich programming environments that show that these environments can effectively be used to learn computer science.

Meanwhile, the answer to the physicality question is less clear. In Study 1, the class that used a combination of virtual and physical VEX robots showed significant improvements in algorithmic thinking, while the classes who used only virtual VEX robots showed marginal gains at best. One possibility is that the physical VEX robot provided students with a stronger association between their programming codes and their robot's behaviours. This association could benefit algorithmic thinking, in that students are more able to understand how the specifics of their programs cause the robot to act; this knowledge may allow them to think through and plan their programs with more accuracy. On the other hand, a virtual simulation allows students to receive quick feedback and to immediately fix their errors and restart their program simulations. Other literature in the programming and computer science education field has emphasized the importance of short code execute debug cycles, which aid the learning process and may also prove less frustrating for students (e.g. Barnes, 2002; Dodds, Greenwald, Howard, Tejada, & Weinberg, 2006; Fagin & Merkle, 2002). Still, studies in other fields suggest that immediate feedback can actually be detrimental (e.g. Kulhavy & Anderson, 1972; Surber & Anderson, 1975). In this context, the quicker feedback and ability to immediate restart one's program may encourage students to engage in trial and error rather than thinking fully through their program and its behaviours.

However, the comparison in Study 2 between the Virtual class and the Physical class showed no differences in algorithmic thinking. Instead, the Virtual class showed a large time benefit, in that the Virtual class was able to learn the same amount of material in a month's less time. The virtual robot was not burdened with the same limitations that the physical robot had, such as mechanical and electrical problems, allowing students to focus their full attention on the RVW programming course. Thus, it appears that the physicality of the VEX robot did not provide any affordances to the learning process, and instead produced issues that distracted from the robot programming course.

Alternatively, it is possible that a more robust difference in algorithmic thinking would be seen when using a combination of virtual and physical robots, as opposed to exclusively virtual robots or exclusively physical robots. Indeed, in several other fields, a combination of virtual and physical manipulatives have been shown to lead to learning improvements above those acquired when using only virtual or physical manipulatives alone (e.g. Jaakkola & Nurmi, 2008; Martínez-Jiménez, Pones-Pedrajas, Climent-Bellido, & Polo, 2003; Olympiou & Zacharia, 2012; Zacharia, 2007; Zollman, Rebello, & Hogg, 2002). In relation to algorithmic thinking, the virtual environment's affordance of quick feedback and error correction could lead to faster learning of basic programming concepts (which is supported by Study 2's results). Subsequent activities that involve the physical robot could then emphasize the connection between the programming and the robot's behaviour, which could ultimately support greater improvements in more abstract algorithmic thinking. This alternative explanation can be elucidated with future studies that contrast classes using a combination of physical and virtual robots vs. classes that use exclusively one type of robot.

Still, there exists the possibility that the different algorithmic thinking gains seen in Study 1 were caused by class-level differences, rather than the use of virtual or physical VEX robots. That is, differences in the way that Class V + P was taught, in comparison to Classes V1 and V2 from Study 1 (e.g. more emphasis on algorithmic thinking outside of the curriculum), may have led to the significant gains seen in Class V + P. Furthermore, the repetition of exercises in Class V + P (first with a virtual robot, then with a physical robot) may have reinforced the algorithmic thinking concepts learned by students (though the other sub-categories did not show evidence of reinforcement). Another possibility is that the pretest and post-test did not provide an accurate measure of algorithmic thinking ability, as the tests included only four algorithmic thinking-related questions that had a mediocre instrument reliability (as evidenced by a Cronbach's alpha of .54). Future studies should address this risk by using an assessment that includes more algorithmic thinking-related questions with a higher consistency, to ensure that the test is accurately measuring the construct of algorithmic thinking.

The current studies focused primarily on learning and time affordances provided by virtual and physical learning environments, but there are other potential affordances that remain to be explored. For example, working with physical robots may feel more authentic for students interested in robotics, which could lead to motivational differences within the two types of environments. In this case, graphically complex, three-dimensional environments such as the RVW platform may prove especially valuable: students may feel equally motivated by the immersive world, fantasy elements and game-like challenges in the simulator, even though they are not working with a physical robot. The use of motivation measures and qualitative observations of student and teacher experiences in future studies would help in studying motivational affordances within virtual and physical environments.

In conclusion, the RVW environment is another example of a rich context that can be used to teach computer science concepts, embedded in the context of robotics. While the current studies did not provide a clear answer as to whether the lack of physicality in the virtual environment affects learning, it is beneficial to consider the unique learning affordances provided by virtual environments and physical environments. Virtual environments provide great advantages in terms of accessibility and convenience for classrooms that may otherwise be unable to participate in robotics programming, due to financial and time costs and physical space limitations. By further investigating the unique affordances provided by physical environments, it may be possible to integrate their unique learning affordances into virtual environments, such that computer science and programming environments can continue to increase in accessibility with no learning affordances lost.

Acknowledgement

The project described was supported by Award Number T32GM081760 from the National Institute of General Medical Sciences.

Funding

This work was supported by the Defense Advanced Research Projects Agency [grant number #FA8750-10-2-0165].

References

Barnes, D. J. (2002). Teaching introductory Java through LEGO MINDSTORMS models. *SIGCSE, 34*, 147–151.

Bishop-Clark, C., Courte, J., & Howard, E. V. (2006). Programming in pairs with Alice to improve confidence, enjoyment, and achievement. *Journal of Educational Computing Research, 34*, 213–228.

Clements, D. H. (1990). Metacomponential development in a Logo programming environment. *Journal of Educational Psychology, 82*, 141–149.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika, 16*, 297–334.

Dann, W., Cooper, S., & Pausch, R. (2006). Learning to program with Alice. Upper Saddle River, NJ: Prentice Hall.

Dodds, Z., Greenwald, L., Howard, A., Tejada, S., & Weinberg, J. (2006). Components, curriculum, and community: Robots and robotics in undergraduate AI education. *AI Magazine, 27*, 11–22.

Fagin, B. S., & Merkle, L. (2002). Quantitative analysis of the effects of robot on introductory computer science education. *Journal on Educational Resources in Computing, 2*(4), 1–18.

Jaakkola, T., & Nurmi, S. (2008). Fostering elementary school students' understanding of simple electricity by combining simulation and laboratory activities. *Journal of Computer Assisted Learning, 24*, 271–283.

Kulhavy, R. W., & Anderson, R. C. (1972). Delay-retention effects with multiple choice tests. *Journal of Educational Psychology, 63*, 505–512.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., … Thomas, L. (2004). A multinational study of reading and tracing skills in novice programmers. *SIGCSE Bulletin, 36*, 119–150.

Major, L., Kyriacou, T., & Brereton, O. P. (2011). Systematic literature review: Teaching novices programming using robots. In *15th Annual Conference on Evaluation & Assessment in Software Engineering* (pp. 21–30). Staffordshire.

Martínez-Jiménez, P., Pones-Pedrajas, A., Climent-Bellido, M. S., & Polo, J. J. (2003). Learning in chemistry with virtual laboratories. *Journal of Chemical Education, 80*, 346–352.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., … Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin, 33*, 125–140.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2010). Learning computer science concepts with Scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research* (pp. 69–76). New York, NY: ACM.

Nunnally, J. C. (1978). *Psychometric theory* (2nd ed.). New York, NY: McGraw-Hill.

Olympiou, G., & Zacharia, Z. C. (2012). Blending physical and virtual manipulatives: An effort to improve students' conceptual understanding through science laboratory experimentation. *Science Education, 96*, 21–47.

Papert, S. A. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York, NY: Basic Books.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM, 52*, 60–67.

Surber, J. R., & Anderosn, A. C. (1975). Delay-retention effect in natural classroom settings. *Journal of Educational Psychology, 67*, 170–173.

Sutherland, R. (1993). Connecting theory and practice: Results from the teaching of Logo. *Educational Studies in Mathematics, 24*, 95–113.

Sykes, E. R. (2007). Determining the effectiveness of the 3D Alice programming environment at the computer science level. *Journal of Educational Computing Research, 36*, 223–244.

Vesgo, J. (2005). Interest in CS as a major drops among incoming freshman. *Computing Research News, 17*, 236–248.

Zacharia, Z. C. (2007). Comparing and combining real and virtual experimentation: An effort to enhance students' conceptual understanding of electric circuits. *Journal of Computer Assisted Learning, 23*, 120–132.

Zollman, D. A., Rebello, N. S., & Hogg, K. (2002). Quantum mechanics for everyone: Hands-on activities integrated with technology. *American Journal of Physics, 70*, 252–259.

Appendix

General programming

- Special types of variables that allow you to pass data into a function are called (senders/parameters/passers/variables).
- Just like task main, functions must have a set of parentheses () and a set of (semi-colons/curly braces/AND operators/brackets).
- If the condition of an If statement is true, then all of the code inside of its curly braces will run. True/False.
- A decimal number can be stored in which type of variable? (int/char/string/float).
- With functions, you write the code once, give it a name and then reuse it as many times as you'd like within a program. True/False.

ROBOTC syntax

- The code "wait1Msec(5000)" tells the robot to wait for (5 s/50 s/one second/half a second).
- Which character is used to signify the end of a command in ROBOTC? (comma/period/semicolon/colon).
- Before using the encoders in a line-tracking program, their sensor values should be set to (−1/4095/0/100).
- Which of the following cannot be used when naming a motor or sensor? (special characters/spaces/ROBOTC reserved words/all of the above).
- To make a robot stop, you set its motor values (equal to 0/opposite one another/equal to 63/equal to −1).

Physical robot functioning

- Ignoring drift, when using encoders to control how far a robot travels, slowing the robot down will make it move (a greater distance/the same distance/a shorter distance/an unknown distance).
- What can cause one motor to run faster than another on a robot? (construction of the robot/variances between the motors/friction/all of the above).
- On the VEX Cortex, the shaft encoders record how many counts or degrees per revolution? (360/100/180/1000).
- The VEX Line Tracking sensor works by measuring (reflected light/torque/sound/

RPMs).

- A properly configured robot that has one motor turned on and the other turned off will perform a (opposite turn/point turn/tank turn/swing turn).

Algorithmic thinking

- The hybrid language halfway between English and the programming language is called (halfcode/hybridcode/prenglish/pseudocode).
- Simple behaviours are made up of complex behaviours. True/False.
- When comparing the role of the robot vs. the role of the programmer, the programmer's role is to (ignore/enforce/create/carry out) the plan.
- Given the program above, the robot will do ___.