Ruggedizing CS1 Robotics: Tools and Approaches for Online Teaching

Boyd Anderson
Martin Henz
Hao-Wei Tee
National University of Singapore
Singapore

Abstract

First-year students benefit from robotics-based programming exercises by learning how to use sensors to gain information on the (changing) world surrounding the robot, how to model this information using data structures, and how to design algorithms for performing meaningful activities. Robotics-based exercises are naturally experiential and team-based and provide among the most memorable teachable moments of first-year programming courses. We summarize the pedagogical challenges that robotics-based exercises face, even under ideal circumstances, and how a university responded to these challenges. We report on the additional challenges faced in late 2020 at the same university as a result of the COVID pandemic, and how the course staff addressed these challenges using programming language implementation and network tools. The crucial components were (1) a custom-built web-based development environment with collaborative features including a built-in compiler, (2) a portable virtual machine, (3) collaborative editing, (4) open source protocols, and (5) peer-to-peer teleconferencing software. We report on the lessons learnt and how to further improve the resilience of robotics-based programming exercises.

CCS Concepts: • **Social and professional topics → CS1**; • **Computer systems organization → Robotics**.

Keywords: educational robotics, teaching CS1 using robotics, online robotics, learning tools

ACM Reference Format:
Boyd Anderson, Martin Henz, and Hao-Wei Tee. 2021. Ruggedizing CS1 Robotics: Tools and Approaches for Online Teaching. In *Proceedings of the 2021 ACM SIGPLAN International SPLASH-E Symposium (SPLASH-E '21), October 20, 2021, Chicago, IL, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3484272.3484969

SPLASH-E '21, October 20, 2021, Chicago, IL, USA

ACM ISBN 978-1-4503-9089-7/21/10.
https://doi.org/10.1145/3484272.3484969

1 Introduction

The epistemological foundation for the use of robotics in education was laid by Jean Piaget [8], the founder of constructivism, who emphasized the importance of the learner's active construction of knowledge. Seymour Papert's constructionism highlighted the value of making tangible objects in the real world in this process of knowledge construction. Papert's book "Mindstorms" [7] developed the foundation for the use of robotics in education. The collaboration of Papert's MIT Media Lab with the LEGO company led to the development of LEGO MINDSTORMS [5]. Since the introduction of this product line in 1998 and other similar products, the use of robotics in education has proliferated and is widely reported in the computer science, engineering and education literature. A recent literature survey [2] lists over 140 studies on robotics in education from 2000 to 2018. The survey also provides a more detailed history of robotics in education.

Today, examples of robotics-based CS1 courses include "MIT: 6.01 Introduction to EECS via Robotics" [6] and "KAIST: CS101 Introduction to Programming" [4]. The CS1 course discussed in this paper follows the textbook *Structure and Interpretation of Computer Programs, JavaScript Edition* [1] (SICP JS), but is taught using a small JavaScript subset called Source, which includes only the features necessary for the textbook. Since 2005, our course has included a robotics component using LEGO MINDSTORMS. The robotics component is introduced right after the students get exposed to imperative programming, following Chapter 3 of SICP JS.

During the COVID-19 pandemic of 2020, all large lectures and the majority of labs and tutorials in our university had to be moved online, using teleconferencing tools in the first semester of Academic Year 2020/21. Travel restrictions forced many of our students to take all their classes remotely. 450 students were required or opted to take their laboratory sessions online. With its aspiration of letting students "make tangible objects in the real world", the robotics component of the course was severely endangered by the COVID-induced restrictions.

Section 2 describes the use of robotics in this course up until 2019 in more detail. Section 3 reports how we responded to the challenges posed by COVID with a combination of technological, logistical, and organizational measures. Sections 4 and 5 describes the robotics setup developed for the

module. Section 6 describes the learning outcomes that we achieved, the observed shortcomings, and future steps to further increase the resilience of a robotics-based component in a CS1 course.

2 Pre-2020 Robotics Course Component

From the beginning, a goal in our CS1 course was to enable all of our students to complete all their assignments in a web-based development environment called the Source Academy. This approach removed the need for first-year students to have to figure out how to set up a local JavaScript development environment, install libraries and deal with operating system or architecture differences. By removing these barriers, the students could focus on learning programming methodology first, and then learn how to use these powerful and complex tools in the future.

However, before 2020, the robotics component of our course was not fully integrated into the Source Academy. The students could write their programs in JavaScript, but running these programs required the setting up of an additional development environment which necessitated the use of SSH and SCP. This initial setup used only off-the-shelf components: the LEGO MINDSTORMS EV3 kit flashed with the ev3dev Linux distribution[1] bundled with Node.js and a small wrapper script that included a customised ev3dev JavaScript API into the Node.js environment.

Although these were all packaged into a single image that students could write directly to a microSD card, students still had to figure out how to connect to the EV3 over SSH to run their programs. If they wanted to write their programs in the Source Academy, they had to manually transfer their programs to the EV3 by copy-and-pasting them from their browser into a file, and then transfer the file over using SCP. They would have to do this for every modification they made to the program. Alternatively, they could write in a text editor (such as vim) over SSH on the device itself, but then they lose the functionalities of the web-based development environment such as code completion. The students also had to be physically present to be able to program the robot, even if they were connecting to it over a wireless network.

In addition to these barriers, by far the biggest issue was the poor performance; the EV3 runs a single low-power ARM core clocked at 300 MHz, which is not sufficient to run Node.js at an acceptable speed. Each time students ran a program, they had to wait about 10 seconds for Node.js to initialise before their program ran. This on top of the manual file transfer provided for a rather frustrating development experience, especially as robotics programming tends to involve a lot of trial-and-error.

Overall, the original setup allowed the students to program the LEGO MINDSTORMS EV3 in JavaScript and experience robotics in their first year. However, these barriers and limitations needed to be addressed to achieve our goal of allowing students to concentrate on programming methodology first. As it happened, these changes would need to occur in the following year.

3 Going Online

During the second half of 2020, our CS1 module had to be run online. All lectures and tutorials were taught using teleconferencing tools due to the large class size (625 students) and social distancing considerations. Our lab sessions however were split between online and face-to-face groups (see Figure 1) as the government COVID-19 guidelines allowed us to have classes of up to 8 students. However, approximately two-thirds of our students attended online sessions.

In addition to this, travel restrictions prevented many of our students from entering our country at all and therefore they were required to take all classes remotely. This meant there could be no physical contact with these students, and therefore all of our assessments and lab sessions needed to be teachable virtually.

Despite these constraints, we were determined to continue to run the robotics component of the module, because the past years clearly showed the pedagogical benefits of a robotics-based experiential component in the course. We were fortunate, in that our efforts to address the limitations of the robotics component, which are covered in the following sections, had helped us prepare a suite of tools for teaching robotics remotely.

There were issues of logistics; for example, we had to make sure that each of our lab groups contained at least one person who was present on campus. This person would become the "Robot Custodian" and would collect the LEGO MINDSTORMS EV3 and printed puzzle sheets, manually build the robot in consultation with their team mates, and work with their lab group on the robotics assessments of the module. Outside of this one student, the rest of the robotics component of the module was handled remotely. The tools required to do this are discussed in the next sections.

4 A Virtual Machine (VM) for Educational Robots

Our first consideration to improve our robotics component was to improve the delay for running student programs. The main culprit is that Node.js is poorly suited to run on microcontrollers. It uses the V8 runtime[2], a heavyweight VM that has a just-in-time compiler, which causes far too much overhead for short-running programs on a low-powered processor such as those on most robot microcontrollers. V8 also has to parse the program on the device itself, which adds to the startup overhead. It implements almost the entire JavaScript language which has many features that are not

[1] ev3dev, https://www.ev3dev.org/

[2] V8 JavaScript Engine, https://v8.dev/

Figure 1. Student in Lab sessions using the EV3 Mindstorm Robots in a Face-to-Face setting and in an Online setting

included in Source, adding further unnecessary overhead as the VM has to be designed to support these features; it, however, does not implement proper tail calls, which is a feature that we rely on, given our CS1 course's focus on functional programming.

Another issue was that Node.js dropped support[3] for the version of the ARM architecture used by the EV3, and so we were limited to Node 0.10, which, among other things, lacked support for arrow functions and `const` declarations, which are used frequently in Source.

Any replacement for Node.js would have to (1) support the features of ES6 used in Source, namely arrow functions and `let` and `const` declarations; (2) be usable from a purely web-based environment, if any compilers or other tools were required; and (3) be small enough to run on microcontrollers.

We considered JavaScript runtimes designed for use on embedded devices. Duktape[4] and Espruino[5] were not suitable as they supported only ES5.1, and we were not keen on adding significant functionality to an unfamiliar codebase—we would have had to modify both the parser and interpreter of the chosen runtime. Moddable's XS[6] was promising, but their toolchain relies on a C compiler, which we cannot provide in a web environment. We therefore decided to write our own runtime. Because Source excludes much of the more complex features of JavaScript, like prototypal inheritance and objects, this was much simpler than implementing a full JavaScript runtime.

We developed SVML (Source virtual machine language), a stack-based virtual machine designed specifically for our use case: as a compilation target for Source that is efficient to run on low-power devices such as microcontrollers. We also wrote a simple interpreter for SVML in C, called Sinter (Source interpreter), and a compiler from Source to SVML, which was written in TypeScript so that it could be used from a web environment. SVML and Sinter together solve the performance issue; programs start running almost instantly, improving the development experience. However, we still wanted to address the barrier of having to copy programs directly to the robot every time a change was made.

5 Web-Based Robot Interactivity

Such a VM setup by itself would not have worked at all during a pandemic as some students were not allowed or able to meet; fortunately, our web-based development environment has a collaborative editing feature that allows students to work on the same program in a manner akin to Google Docs. Our collaborative editor uses ShareDB, a realtime database built on operational transformation [3] of JSON documents.

Had we gone into the pandemic with the pre-2020 robotics setup, students could have used the collaborative editor to work on their programs together, but the student who physically had the robot would have had to manually transfer the program to the EV3 and run it each time anyone wanted to try a program; it likely would have worked, but it would not have been a good experience.

To make the development experience truly seamless, we wanted to allow programs to be run directly from the Source Academy, without students having to exit the development environment at all.

We developed Sling, an MQTT[7]-based protocol that allows programs to be sent from a client to a device, and the output to be sent back from the device to the client. By virtue of MQTT being a publish-subscribe protocol, multiple clients can be connected to a device at a time; they will all see the device's status (whether it is running a program) and output from the program. Any connected client can send a program to the device to be run.

[3] V8 dropped support for architectures without hardware support for floating-point numbers in version 3.18, which Node.js started using from version 0.11

[4] Duktape, https://duktape.org/

[5] Espruino, https://www.espruino.com/

[6] Moddable, https://www.moddable.com/

[7] MQTT is a lightweight publish-subscribe network protocol commonly used in embedded and IoT applications

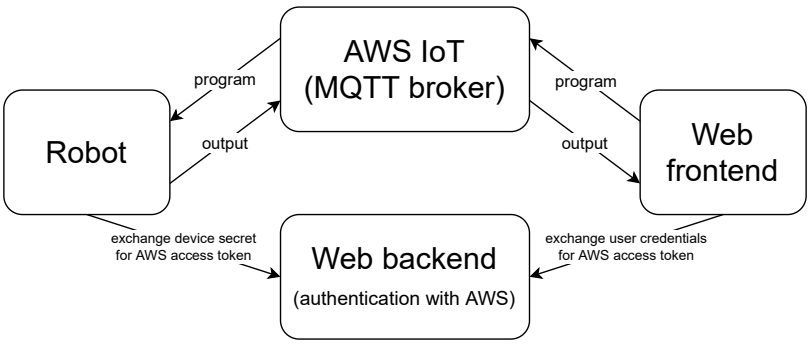

Figure 2. Remote execution flow

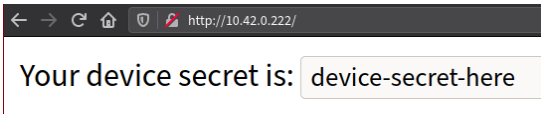

Figure 3. Page showing device UUID

Our implementation of Sling involves four parts: the remote execution host on the device, which receives programs—in our case, compiled SVML programs—and invokes the interpreter, and then relays output back; the client library in our web environment; AWS IoT Core, which we use as our MQTT broker; and a web backend, which handles authentication to the MQTT broker for both clients and devices. Figure 2 depicts the overall flow of the setup.

With SVML, Sinter, and Sling, our students now have a much smoother experience. As in the original pre-2020 robot setup, the students download and flash a preconfigured ev3dev image, which now contains Sinter and Sling instead of Node.js. Upon powering on the device and connecting it to their computer, instead of SSHing onto the device to transfer programs, they can simply open a web page on the device that displays a UUID (Figure 3). They register this UUID to their accounts in the Source Academy (Figure 4), which authorises the device to connect to the MQTT broker, and authorizes them to connect to the device. They can then simply select the device as an execution target (Figure 5), and then program in the Source Academy as per normal. Each time, any team member hits the "Run" button in the web-based programming environment, their program runs on the robot.

If students are not able to meet in person, they can separately connect to the same device and collaborate over our web-based development environment. In practice, the student who physically has the robot would simply point a camera at the robot over a video call with their lab mates. That student no longer has to repeatedly manually transfer programs to the device, and the students can instead concentrate on the course content. Combined with the collaborative editor, students can work on the same program and see the

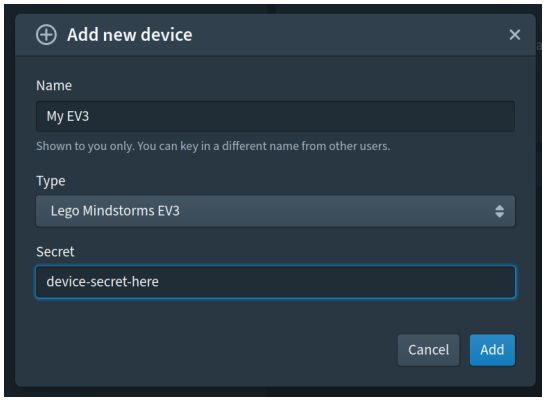

Figure 4. Registering a device

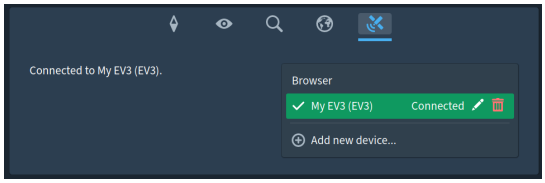

Figure 5. Selecting a device

same output from each execution in their own browser. This combination of tools allowed students to collaborate on the robotics component of the module when working side by side, remotely, and even across borders.

6 Lessons Learned

When everything is "business as usual", it is easy to make assumptions about the state and suitability of our pedagogical tools. A setup might be "good enough" when the majority of students don't have trouble with getting started with the robot, or when other students or teaching assistants can help. Collectively a cohort can reach the desired outcome without too much overhead. An instructor can also intervene to course-correct if there are some impediments to learning. However, disasters such as COVID-19 change this picture; these small nuisances can become show-stoppers. One extra

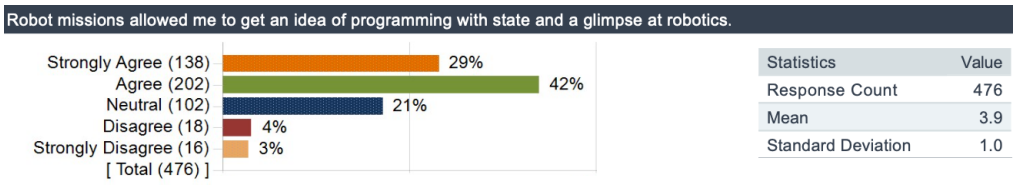

Figure 6. Post-course Feedback on Robot Component

step can become a step too many and the teaching process breaks down.

We were worried that the robotics component of our CS1 module was especially at risk for these sort of setbacks and barriers. Our original setup required too many additional components and steps. Therefore, our goal was to eliminate these unnecessary steps and let the students focus on their learning, regardless of the device they are using or their location. We believe we were successful in this goal, and during our 2020 online semester, we were able to run our robotics component with positive student feedback (see Figure 6).

We were fortunate to have some crucial pieces of our setup in place before disaster hit. Our open-source, web-based environment, developed and maintained by a team that knew it well, allowed us to adapt our setup quickly. We also already had an existing language implementation setup with parsers and programming environment. In addition, as we only had to support a minimalistic programming language (Source, our JavaScript subset), we were able to build a small and portable virtual machine and interpreter. This would not have been possible if we had to support all of JavaScript's functionality. Another important aspect was the robot kits we had access to: the LEGO MINDSTORMS EV3 kit permits open-source development (ev3dev) allowing us to run our virtual machine on the device.

We were able to focus on the deployment of student programs from the browser to the EV3 platform. We decided for a VM-based approach as a good compromise. It allowed a fairly straightforward robot programming workflow. The student program is compiled in the browser to our virtual machine language, and then is sent to the device over the internet, and then run on a dedicated virtual machine running on the EV3. The focus on this limited subset of JavaScript ultimately made it possible to use this VM approach and therefore ruggedize the course.

We see this setup as a framework to grow and improve the robotics component of the course. The development on the virtual machine interpreter has not stopped. Since the 2020 semester, it has been ported to the ESP32 and Arduino platforms. This opens up new possibilities for programming more than just the EV3 robots. Indeed, we are in the process of prototyping an ESP32-based robot and investigating controlling drones with our setup. We see many exciting possibilities for teaching with robots, from letting students experience distributed programming with multiple connected and programmable devices to allowing students to coordinate multiple robots together.

7 Using Sinter and Sling

The Source Academy[8], our Source language implementations and SVML compiler[9], Sinter[10] and Sling[11] are all open-source projects under the Apache license, and are available for any educators to experiment with and adopt in their own courses.

Sinter is written with portability and extensibility in mind: it is easy to port Sinter to any platform with a C11 compiler and extend the VM so that programs run in the VM can interact with features of the platform, including doing I/O and controlling peripherals like sensors and motors.

Our Sling implementation will run on any Linux-based platform. As network programming interfaces differ from platform to platform, using Sling on a new platform will likely require a new implementation of Sling.

References

[1] Harold Abelson and Gerald Jay Sussman. 2022. *Structure and Interpretation of Computer Programs* (JavaScript ed.). MIT Press, Cambridge, MA. Adapted to JavaScript by Martin Henz and Tobias Wrigstad, ISBN: 978-0262543231.

[2] S. Anwar, Bascou N. A., M. Menekse, and A. Kardgar. 2019. A Systematic Review of Studies on Educational Robotics. *Journal of Pre-College Engineering Education Research* 9, 2 (2019), 19–42. https://doi.org/10.7771/2157-9288.1223

[3] C.A. Ellis and S.J. Gibbs. 1989. Concurrency control in groupware systems. *ACM SIGMOD Record* 18, 2 (1989), 399—-407. https://doi.org/10.1145/67544.66963

[4] Korea Advanced Institute of Science and Technology. 2021. CS101: Introduction to Programming. (April 2021). http://web.archive.org/web/20210412213143/https://cs101.kaist.ac.kr/.

[5] LEGO System A/S. 2021. LEGO MINDSTORMS. (September 2021). http://web.archive.org/web/20210910151954/https://www.lego.com/en-us/themes/mindstorms.

[6] Massachusetts Institute of Technology, Electrical Engineering and Computer Science. 2021. 6.01 Introduction to EECS via Robotics, Spring 2021. (February 2021). http://web.archive.org/web/20210203200039/http://student.mit.edu/catalog/m6a.html.

[7] S. Papert. 1980. *Mindstorms: Children, computers, and powerful ideas.* Basic Books, Inc., New York. ISBN: 978-0465046270.

[8] J. Piaget. 1970. *Genetic epistemology.* W.W. Norton & Company, New York. ISBN: 978-0231033862.

[8] Source Academy, https://github.com/source-academy
[9] js-slang, https://github.com/source-academy/js-slang
[10] Sinter, https://github.com/source-academy/sinter
[11] Sling, https://github.com/source-academy/sling