Reducing Instructor Workload in an Introductory Robotics Course via Computational Design

Devon J. Merrill
UC San Diego
devon@ucsd.edu

Steven Swanson
UC San Diego
swanson@ucsd.edu

ABSTRACT

Physical computing and building robots has important benefits for novice engineers and computer scientists. However, lab time and hardware debugging comes with a high cost of instructor time and effort. To reduce this workload, we implemented a computational design tool that simplifies printed circuit board (PCB) design and manufacture, assembly, and programming. We pilot tested our computational design tool in a one-unit introductory physical computing course for 196 CS1 students. The students designed, assembled, and programmed a custom robot with minimal instructor assistance. The robots are Arduino-based and each included a student designed PCB. The students assembled the robots from off-the-shelf electronic components according to automatically generated assembly instructions. Students programmed their robots using simple APIs that were automatically generated and customized for each unique robot. A minimum of two quarters after the completion of the course, grade point average (GPA) for the students who completed the course, was found to be 0.15 higher than a comparison group of similar student ($n = 498$, $p < 0.05$). We present a detailed description of our computational design tool and course curriculum, identify challenges encountered by the students and instructional staff, make recommendations to increase student achievement, and address the scalability of the course.

ACM Reference Format:
Devon J. Merrill and Steven Swanson. 2019. Reducing Instructor Workload in an Introductory Robotics Course via Computational Design. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287506

1 INTRODUCTION

A diverse range of computing devices are becoming deeply embedded in almost all aspects of our day-to-day lives. The increasing importance of computing and its growing economic impact are driving enormous growth in undergraduate demand for computing courses. These two trends lead to challenges in providing large numbers of students with experiences that straddle the boundary between software and the real world. Furthermore, growing enrollments often lead to large undergraduate classes that can be impersonal and make it difficult to have effective small-group interactions.

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02.
https://doi.org/10.1145/3287324.3287506

We have developed Robot Parade, a lightweight lab course for freshmen computer science students taken concurrently with CS1, usually during their first quarter at college. We built the course to show students that they can conceive, build, and program a complete computing device of their own design. Robot Parade aims to provide a low-stress, enjoyable venue for students to practice what they learn in CS1 while interacting with other students in a smaller, more intimate setting than the large lecture format of our CS1 course can offer. Students complete nearly all the coursework in class during weekly two-hour labs, held nine times during the quarter.

Using robots in introductory programming courses can improve attitudes toward computers in general [4], and this approach has been developed as a direct application of Piaget's constructionism learning theory (e.g., by Rosenblat and Choset [5]). However, teaching these courses can be resource- and time-intensive. Rosenblat's course, for instance, relied on a low student-to-instructor ratio and pairing experienced senior students with less advanced students. These problems are especially acute for courses targeted at less-experienced students.

We developed Robot Parade to make a hands-on programming and building experience accessible to students during their first term studying computer science in college. A key motivation for targeting students early is to reduce attrition among underrepresented groups of CS and CE majors.

The course centers around a single project: students design, build, and program a simple wheeled robot that can sense and react to the environment. The project includes creating and assembling a custom printed circuit board (PCB).

Enabling first-term students who are learning to program and have little (or no) experience with electronics to complete the project has required us to carefully craft the course's content and schedule and develop a robot design tool that is accessible to our students.

During the first half of the course, students complete several robot programming labs using pre-built robots that include all the elements they will eventually use in their own designs. Each lab focuses on a different aspect of sensing or control (such as driving, controlling lights and a speaker, using a distance sensor). These labs help students learn the basics of Arduino-style microcontroller programming, and grapple with the vagaries of using code to gather data from sensors and control physical devices. They also encounter the rudiments of C++ (we teach CS1 in Java), gain experience assembling electronics (i.e., soldering and wiring), and learn their way around the "Makerspace" where we teach the class and some of the tools (e.g., 3D printers) it provides.

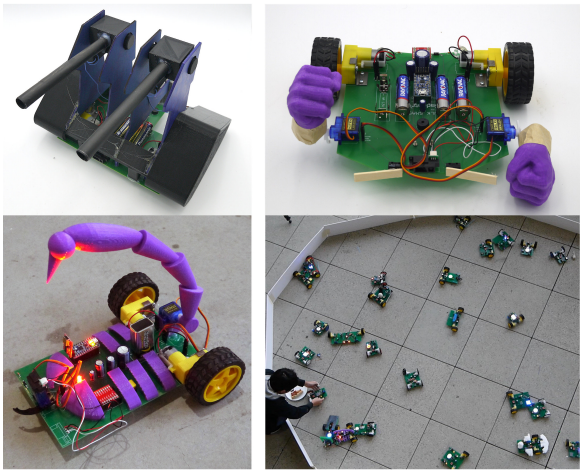

Figure 1: Several student designed robots.

Next, they design a robot to carry out a task of their choosing. "Tasks" include things like "look and act like a duck," "be polite to other robots," or "drive around and pop wheelies."

The key component of the robot is a printed circuit board (PCB) that will host its electronics and serve as its body. Designing PCBs is a complex task that usually requires experience with electronics and the steep learning curve of conventional PCB design tools.

To make PCB design tractable for our students, we have built a web-based, what-you-see-is-what-you-get (WYSIWYG) robot design tool called *The Robot Factory*. The Robot Factory lets students design a working robot in a single class period with very little training. Behind the scenes, The Robot Factory uses computational design to algorithmically generate a PCB that implements the circuitry for their design.

We have their PCBs manufactured, and the students solder them together by following customized assembly instructions for their robot that The Robot Factory generates. Students program the robots with the Arduino IDE using a customized library that The Robot Factory generates to match their particular robot design.

We have taught Robot Parade in five academic quarters over the last three years. Space is limited primarily by instructor "bandwidth," so we select students by lottery from those that submit a simple application. We typically run two sections of 20-30 students working in pairs. We have been working steadily to increase the number of students that may enroll the course by streamlining our curriculum and refining our robot design tool.

We recruit students who concurrently take a CS1 course alongside our robot course. The majority of the participants do not initially have basic programming skills such as knowledge of variables, functions, and control structures.

The following sections describe The Robot Factory and its interface (Section 2), detail the course content (Section 3), provide a pilot evaluation of the course's impact on student academic success (Section 4), and discuss plans for future improvements to the course (Section 5). Section 6 draws conclusions from our work.

2 THE ROBOT FACTORY

We want students to have the experience of building a real, working robot without requiring them to scale the steep learning curves that hardware design tools usually require. To make the task tractable and allow the students to focus on the *design* of the robot rather than the details of its implementation, we developed a computational design tool called The Robot Factory that makes designing a robot easy and automates much of the implementation.

The Robot Factory caters to students that have no electronics design or programming experience, and it minimizes the amount of instructor assistance and prompting required for success. To achieve both of these goals at once, we built it to be simple to use, provide real-time feedback about potential design problems, and use a familiar "drag-and-drop" paradigm for design. As a result, students can create a valid robot design after just a short two-minute introductory demonstration.

The Robot Factory also generates an easy-to-use high-level application programming interface (API) for each robot. The API uses concepts familiar to students taking CS1 so programming the robots is relatively easy as well.

Below, we describe the tool's interface and give an overview of how it works internally. It is available for anyone to use (http://robots.gadgetron.build/).

2.1 User Experience

The Robot Factory provides a web-based, drag-and-drop GUI that students use to design the shape of their robot and the components it will comprise. When it opens, it presents students with a minimal robot design on a green polygon that represents the PCB. The minimal design includes a ProTrinket [3] Arduino board and four AA-size batteries.

Students add *functional components* to the PCB from a library. They can choose from motorized wheels, buttons, single LEDs, LED arrays, IR distance sensors, actuated claws, bump switches, and a few other items.

The Robot Factory automatically adds *utility components* that the robot must include to work properly and synthesizes the PCB circuits to connect all the components together. Utility components include resistors and capacitors, motor drivers, and protection diodes. Large utility components that will significantly affect the appearance of the robot (e.g., the batteries and large capacitors) are visible in the user interface so students can choose where to place them. The tool automatically places small utility components (e.g., resistors).

After students are satisfied with their design they press the "build" button. This button sends the robot specification to synthesis servers running in the cloud.

Placement errors. The locations of the components are subject to several constraints that the student designs must satisfy (the wheels must be off the board, parts cannot overlap, etc.). Typical PCB design tools provide a separate design rule check (DRC) command to check some of these constraints, but the output can be confusing, the tools tend to generate many redundant errors, and it can take several iterations to resolve all the problems. Furthermore, some errors can safely be ignored, but knowing which ones are spurious requires expert knowledge.

The Robot Factory makes it easy to design correct robots with real-time feedback about placement restrictions. Difference colors indicate each of the three types of constraint (must be on the board,

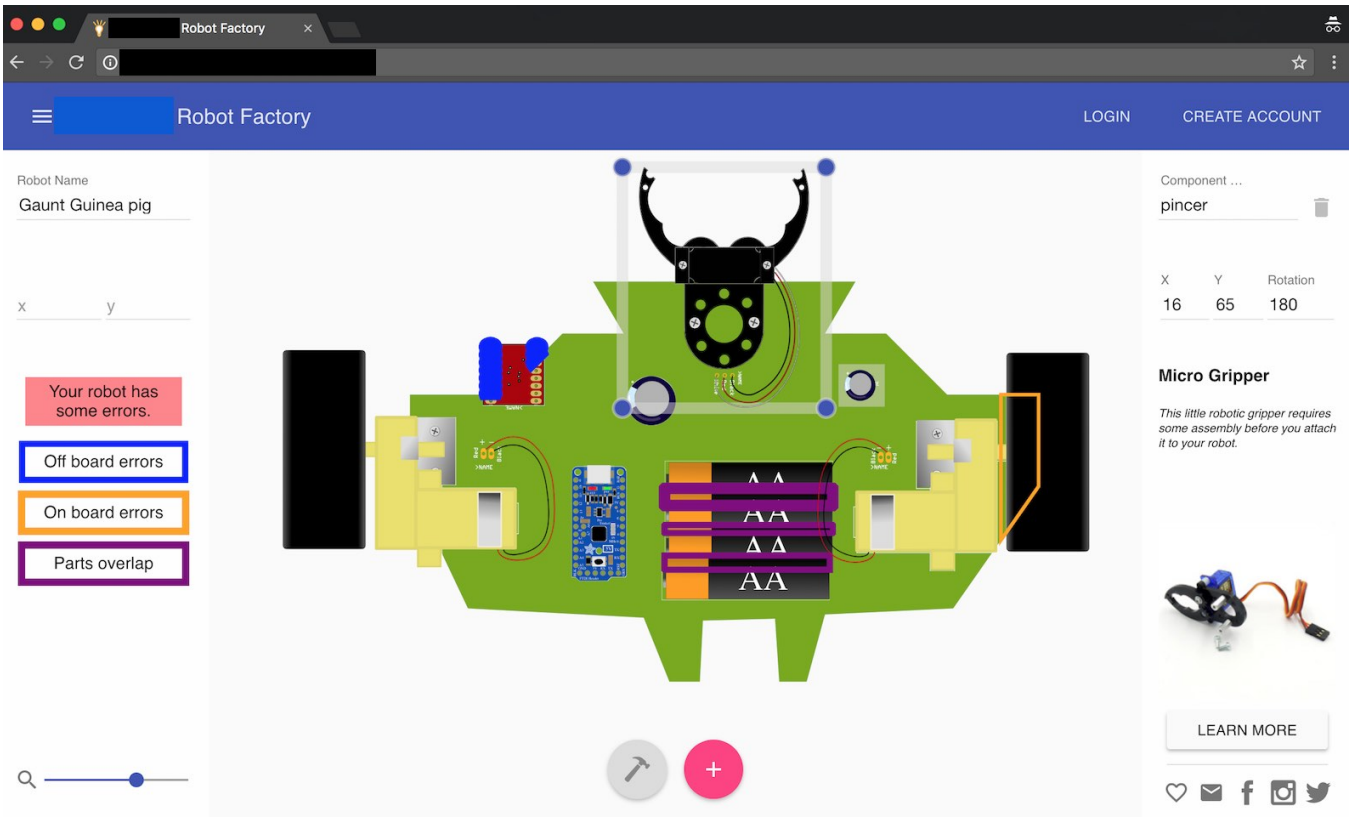

Figure 2: The computational design tool GUI.
An in-progress robot design showing several types of placement errors.

must be off the board, and no overlap). When a constraint is violated, the area of the component that violates the constraint is highlighted with a pulsing, outline. The first time each type of error is detected, a pop-up window explains the error and how to resolve it.

If an error exists, a prominent indicator in the GUI informs the user "Your robot has errors." If no component has a placement error, the indicator becomes green and reads, "Your robot looks great!" Students can only submit error-free designs.

To ensure students learn to deal with errors quickly, we place the components of the initial design so they overlap. Students typically respond by resolving this placement error as their first interaction with the tool. This seems to facilitate student willingness to experiment with component placement and resolve errors on their own.

Board shape. Letting students express their creativity is an important part of the class, but our initial implementation of the tool restricted students to rectangular boards, limiting the range of designs. Since then, we have added support for more complex shapes.

When we first introduced the ability to create complex robot shapes, we noticed that almost every student-created robot still had a rectangular board. We believed that this is because the starting board shape was rectangular. In the latest iteration of the tool, we randomized the vertices of the starting polygon. This has led to a much larger diversity of student designs. For instance, one team designed their board in the shape of the letters "LA" and then programmed the robot to dance to music from the movie "LA LA Land."

2.2 Back-end

The Robot Factory must be as automatic as possible to support scaling the number of student that we can support in a single class. In the vast majority of cases, translating a student's design into the hardware and software required to implement it must not require any direct intervention from the course staff. Furthermore, it must, to the extent possible, guide the students through the process of assembling and programming their robot.

The Robot Factory synthesizes a complete manufacturing specification from the students' designs. This includes PCB computer aided manufacturing (CAM) files, C++ API files, assembly instructions, and bill of materials. The complete process takes between one and five minutes, with the majority of that time spent on auto-placement and routing for the circuit board.

Synthesizing a complete manufacturing specification from an incomplete graphical specification is a multi-step process. We discuss each of them below.

Component-level synthesis. Component-level synthesis makes the necessary electrical connections between the functional components in the design and adds utility components that are not visible in the tool's interface. For example, students add motorized wheels to their design and choose where to put them, but the motors require more current than the microcontroller can provide. In response, The Robot Factory automatically adds and connects a motor driver board and a large capacitor. The motor driver board requires several GPIO connections to the microcontroller, logic level power, and motor power, so The Robot Factory connects (and adds if not already present) batteries and the microcontroller.

Some robot designs are not synthesizable. For example, if the student adds 20 LEDs, but the microcontroller only has 10 GPIO pins available, not all of the LEDs can be connected. Our tool reports this error to the student. It lists of all components that use GPIO pins and suggests removing some of them.

We provide guidelines to the students so these errors occur infrequently, but the details of these connection constraints are complex. For example, many of the Arduino's pins are multi-function and some functions require access to limited microcontroller resources such as timers. We encode these constraints in a generalized, abstract interface specification for each component. Using heuristic search, The Robot Factory is able to perform component-level synthesis on student robot designs in less than 30 seconds.

Auto-placement. Components must be placed on the PCB without violating their placement constraints. The Robot Factory real-time feedback ensures that the students have already correctly placed the functional components and the utility components that are visible. The Robot Factory adds additional utility components and must find places for them on the PCB. A general solution to the placement problem is not feasible (it is at least NP-complete), but our robot designs are not space-constrained, so a simple "first fit" is appropriate and usually completes in under 15 seconds. If auto-placement cannot find a valid placement, The Robot Factory returns an error and asks the student to make more open space in their design. The auto-placer has never failed in practice.

Electrical auto-routing. The routing process translates the logical connections created during component-level synthesis into metal traces on the PCB.

PCB routing can be very challenging. As with placement, however, the relative simplicity of The Robot Factory designs lets us successfully automate the process. The Robot Factory uses Autodesk Eagle's autorouter.

If a design fails to route, synthesis aborts and the student is prompted to try an alternate placement of their components. Out of the more than 100 designs students have created with our tool, fewer than five have failed to automatically route.

Source code generation. Students in the course are concurrently taking a first programming course, taught in Java. The robots we generate are Arduino-based, so they must program in (a simple subset of) C++. Java and C++ are similar enough that students are capable of creating simple programs for their robots. We typically spend about 15 minutes discussing the differences between Java and C++, and students do not usually struggle with the transition.

The Arduino API presents a larger obstacle. Arduino provides a low-level pin-oriented API that can read and write digital and analog values to and from pins on the microcontroller. The link between these low-level operations and higher-level robot behaviors is not always simple. For instance, driving forward requires manipulating several pins at once.

To streamline the process of programming their robots, The Robot Factory provides a "starter" program built specifically for each robot. The program runs "out of the box" on their robot and tests the operation of each component. It flashes any LEDs, drives forward, etc. The starter files also instantiate a C++ object for each functional component in the design with a name that corresponds to the name of the component the student specified in GUI. These objects support simple, intuitive methods such as LED.turnOn(), LED.turnOff(), wheels.forward(), and wheels.turnLeft().

These starter programs are very effective in reducing instructor workload in class. When used without modification they quickly identify malfunctioning components. This speeds up troubleshooting tremendously because students can accurately report and demonstrate issues.

Also, if a robot is not working when loaded with student code, students tend to blame the robot's hardware. By reloading the starter code, they can retest the hardware quickly. If the test code works, they know it is a problem with their code and they usually attempt to fix it themselves.

Assembly Instructions. Since each robot is different, the process of assembling them varies. To help students assemble their robots, The Robot Factory provides customized assembly instructions as a web page that includes an illustrated parts list, shows them where each part goes, and provide instructions for assembly. The instructions also include videos illustrating basic soldering techniques.

2.3 Design management

Student teams create an account in our tool that includes a team name, password, recovery email address, and association with a class section.

When a student's design is finished, the design is saved and becomes visible to instructors. When the students are satisfied with the design, an instructor confirms the design and it is ready to be manufactured. When all the designs are ready, The Robot Factory generates a combined bill of materials and makes the PCB design files easily available for ordering.

3 COURSE CONTENT

Robot Parade has two phases. In the first, students program robots similar to those they will eventually build. They also learn what is feasible with the components that The Robot Factory provides. In the second phase, they use this knowledge to guide the design of their own robots and then assemble and program them.

We teach the class in a makerspace. Students use 3D printers and other tools to create decorations for their robots and soldering equipment for assembly. We encourage students to use other equipment or bring in materials for decoration. The only out-of-class homework assignment in the course is to 3D something by the end of the fourth week. This helps engage them with the tools in the makerspace.

3.1 Grading

Robot Parade is meant to be a low-stress, fun opportunity for students to practice programming and broaden their horizons within computer science. The grading structure reflects this goal. The course is "Pass/Fail" and does not affect student GPA. We record attendance and the completion of the lab assignments. We use these records to identify students who may need additional coaching and encouragement, not for assigning grades.

3.2 Introductory Labs

The course begins with two introductory labs in which the students program two pre-designed robots to complete a variety of tasks. The labs are spread across three to four class meetings (a total of 6-8 hours). Groups work at their own pace with some encouragement from the course staff to stay on schedule.

The first lab involves driving the robot around on paper and using a marker to trace the path the robot has taken. This directly connects to a "turtle graphics" assignments they are working on in their concurrent CS1 course and that is common as an introduction to both procedural and object-oriented programming [2]. Students are eager to engage in the familiar task and quickly connect with the platform.

Drawing with a robot and drawing with a virtual "turtle" provide vivid lessons about the differences between programming the digital and physical worlds. For example, turning the robot precisely is hard while turning a turtle precisely is trivial.

The major tasks assigned in the first lab are to "draw your favorite shape" using the robot and marker, and to program the robot to "sing and dance" using LEDs and a small speaker on the robot.

Both of these assignments are freeform and credit is given for any attempt. However, if there is time left in class, we encourage them to do more. Although the students know that they will receive the same mark for any attempt, they become invested in the free-form assignments and often create elaborate programs.

The second lab involves interacting with the environment using a distance sensor and a motorized gripper. Students program the robot to maintain a constant distance from a moving object, "explore the room without getting stuck," and locate, grasp, and move a 3D-printed object. Students are told that they must use a state machine programming pattern to complete the final task, and we preface the lab with a brief lecture about the state machines and how they apply to robots.

We have taken steps to reduce the dependence of students on course staff during labs. The most helpful of these has been to put together a troubleshooting list. This list contains seven solutions to common problems that students have. These problems are:

(1) Plugging the programmer in the wrong orientation
(2) Not plugging the USB cable in (properly or at all)
(3) Selecting the incorrect serial port
(4) Selecting the incorrect Arduino board type
(5) Selecting the incorrect programmer type
(6) Using the wrong version of the Arduino IDE
(7) Having a dead battery

We have found that if the student's trouble is not on this list, then the problem generally cannot be solved quickly in class and the student should move to another workstation or switch to a different robot.

By using this list we have been able to focus the majority of in-class instructor effort on coaching program design and encouragement. However, the students need frequent reminders to check the list if they have trouble.

3.3 Robot Design

We weave the concept of creating a robot into the class from the beginning, culminating in "design day" when students design robots with The Robot Factory.

During the introductory labs, we encourage students to come up with a concrete, specific concept for their robot. This leads to the more interesting designs. Some examples of successful concepts include Polite Bot (which would drive around a tip a 3D printed top hat at other robots), Duck Bot (which acted like a duck), Smash Bot (which would wave around a big pair of 3D-printed fists. See Figure 1, top-right), R2-D2 bot (which looked like R2-D2 from Star Wars), etc. Concepts like "it's going to drive around and grab things" lead to less interesting designs and less student engagement.

The bulk of the design process takes most teams 60-90 minutes, but many teams spend the full two-hour class period tweaking their design. By the end of the class period, all the student designs are ready to be manufactured.

3.4 Robot Assembly Logistics

The next step is manufacturing the PCBs and assembling the robots. We order the boards from Advanced Circuits [1], quick-turn PCB manufacturer that will make boards for students for $33/board. The boards take at least seven days to arrive, so there is effectively a 2-week delay between design and assembly. The students spend the intervening class period working on 3D-printed decorations for their robots.

We have engineered the robots so that the most expensive components (the distance sensor, servo motors, drive motors, microcontroller, and LED arrays) are reusable. They all have "headers" that plug into sockets that students solder to the board. The components that are soldered to the board (resistors, capacitors, diodes, and so on) are disposable. Altogether, it costs about $50 to build each robot.

3.5 Robot Assembly in Class

Assembling the robots requires basic soldering as well as screwing and hot-gluing some components together. Most students have never assembled PCB or soldered before.

Assembly proceeds in several stages. These steps typically consume between one and two class periods.

Lecture We give a very brief lecture about the assembly process (described below) and explain the notion of polarized components (that is, components with a symmetrical arrangement of pins that must be inserted in the correct orientation) and how to tell if they have them assembled correctly.

Part collection Before class, the course staff sets out trays with all the components the students will need. The trays are arranged in the same order that the parts appear on illustrated parts lists for their robots and are labeled with the same pictures. Students take

a small plastic bin, form a line, and move down the row of bins collecting their parts.

Dry fitting Students "dry fit" the components to their PCBs by using tape and inserting and bending the wire leads to hold them in place. A member of the course staff checks that everything is correct.

Solder lesson An instructor gives a quick soldering lesson to between one and three groups at a time. It covers how soldering works, what a good solder joint looks like, and a demonstration of how to solder each kind of connection they will need to make on their board.

Soldering! Students solder all the components to their PCB.

Solder check and final assembly Course staff checks each soldering job. If it looks good, the students trim the leads on the components. Then, they plug in the microcontroller and other reusable parts. They have a robot!

Testing The last step is to run the test program that The Robot Factory provides. If something does not work, the course staff helps them resolve the issue.

3.6 Robot Programming

The rest of class time in the course (1-2 weeks) is allocated to programming, decorating, and fine-tuning the software for the robots. We coach the students to use a state machine pattern to enable responsive interactivity. Because students have spent several weeks in the beginning of the course programming similar robots, they are generally able to program their own robot without assistance.

We notice an over-reliance on delay statements for program coordination (similar to other observations of early programmers [7]). We are continuing to develop strategies to help students be successful in creating real-time interactive programs in an embedded environment.

4 EVALUATION

We have taught the course in five academic quarters, Fall 2015, Spring 2016, Fall 2016, Winter 2017, and Fall 2017 and performed a preliminary analysis of how taking the class impacts student success over time. Between Fall 2015 and Fall 2017, 498 students applied to take the course. We admitted students from the applicant pool using a random lottery.

Our analysis shows that completing Robot Parade resulted in a statistically significant increase in overall GPA (Welch's t-test p=0.0011). The mean, cumulative GPA of students who completed the course, as of Summer 2018, is 3.43 (196 students, standard deviation 0.45 points, 4.0 scale). The mean GPA of the comparison group of students who applied to take the course, as of Summer 2018, is 3.29 (302 students, sd=0.56). The difference between the means is 0.148 points.

This pilot analysis indicates that completion of Robot Parade resulted in improved academic performance, and we are encouraged to continue developing the course. A full analysis, including the effect on major retention and with regards to gender is in progress.

Anecdotally, students enjoy designing, building, and programming (and decorating) their robots a great deal. Students attribute unique personalities to their robots, and to other students robots. Many students create elaborate stories about their robots' motivations and behaviors. We have observed that these student storytelling behaviors seem to be absent or strongly attenuated in similar first-year seminar robotics classes where all students are given identical robots. Students "owning" the robot that they designed and built themselves seems to be very engaging and motivating, echoing the findings of previous work [6]. Students often express a desire to keep and to continue working on their robot after the course is completed.

Also, because of the PCBs that forms the core of the robots, students feel that their robots are "real" electronics, as opposed to bread-boarded electronics or toolkits such as Lego Mindstorm. The concept of students designing and building "real" embedded devices seems to be empowering and motivating to the students.

5 FUTURE IMPROVEMENTS

Although the Robot Parade class and The Robot Factory design tool have met their original technical goals, we will continue to evolve the course with two further ambitions in mind.

The first is to continue increasing class size. To date, we have increased the scale of the class from 5 robots per class to 15 robots per class without adding course staff. We have plans to scale the course to 60 students (30 robots) per section in the next iteration (Fall 2018). Scaling the initial labs to that size should be easy, but we expect that efficiently assembling that many robots will become challenging.

One option would be to use solder-free assembly. To achieve this we could build (or buy) a standardized robot board and have students attach it to a laser-cut frame that would correspond to the robot shape they designed with The Robot Factory. They would connect electrical components with "hookup" wire. The resulting robots would be cheaper and easier to assemble, and it would open up more time in the course for programming, but we are worried they would be less engaging and exciting for the students.

Our second ambition is to integrate the course more deeply into our CS curriculum. Currently, the course is optional and loosely coupled to the introductory CS courses. Coupling it more tightly or expanding it to be an alternative format for introductory computing, perhaps in conjunction with our robotics program, would be very exciting.

6 CONCLUSION

The Robot Factory makes it possible for Robot Parade to give students hands-on experience building working robots early in their computer science careers. Of equal practical importance, The Robot Factory makes it easier for student to guide themselves through key parts of the design process, facilitates debugging, and eases the burden on course staff of organizing and ordering robot components.

The feedback from students and the growing demand for the course shows that there is a great appetite for this kind of class among CS students. Our preliminary data suggest it may have a positive impact on their long-term success. We plan to continue refining the class and exploring how to increase the benefits to students.

REFERENCES

[1] Advanced circuits. www.4pcb.com/.

[2] Michael E. Caspersen and Henrik Baerbak Christensen. Here, there and everywhere - on the recurring use of turtle graphics in cs1. *Proceedings of the Australasian conference on Computing education - ACSE 00*, 2000.

[3] Adafruit Industries. Pro trinket 5v 16mhz. https://www.adafruit.com/product/2000.

[4] Seong-Won Kim and Youngjun Lee. The effect of robot programming education on attitudes towards robots. *Indian Journal of Science and Technology*, 9(24), 2016.

[5] M. Rosenblatt and H. Choset. Designing and implementing hands-on robotics labs. *IEEE Intelligent Systems*, 15(6):32–39, 2000.

[6] Yuan Sun and S. Shyam Sundar. Psychological importance of human agency how self-assembly affects user experience of robots. *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2016.

[7] David Weintrop, Alexandria K. Hansen, Danielle B. Harlow, and Diana Franklin. Starting from scratch: Outcomes of early computer science learning experiences and implications for what comes next. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, pages 142–150, New York, NY, USA, 2018. ACM.