Development of a Python-based Platform for Teaching Computer Science

Dawit Girma, Bernard Yett, Nicole Hutchins, and Gautam Biswas

Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37212

KEYWORDS. Robotics, computer science, education, Python, Roboscape

BRIEF. This project consists of the creation of a robotics platform built on Python and a curriculum for middle and high school students in order to increase the amount of students interested in computer science.

ABSTRACT. In 2015, there were 10 times the amount of jobs for computer science (CS) graduates than there were people to fill them. As a result, there has been an increase in CS education efforts for middle and high school students. Current systems such as NetsBlox peak curiosity in students but are based on block-based programming languages (BPLs). While BPLs have proven to be effective for introducing students to CS, for a student to pursue CS, text-based programming languages (TPLs) need to be taught as well. Therefore, a platform that advances from BPLs to TPLs is necessary. The design of this platform used robotics as a medium and included an agent that would send commands to the robots, a server that would facilitate communication between a robot and an agent, and a robot that would perform actions on command. A curriculum was also created in order to see the computational skill gains before and after using the robotics platform. When analyzing the results of the pre- and post-tests, it was shown that the differences between the tests were not significant. However, scores did improve for two students and no scores decreased, demonstrating that this platform can be used for CS education.

INTRODUCTION.

Due to the technological advances of the last two decades, the market for computer science jobs has been slated to increase by 21% from 2018 to 2028 (1). However, this market is highly unsaturated due to the lagging increase of computer science (CS) graduates. According to data from the National Center for Education Statistics, in 2015, there were almost 10 times the amount of jobs available in the market than positions filled by graduates (2). As a result, there has been a push towards teaching computational thinking (CT) skills to middle and high school students through different curriculums, such as Code.org and the AP Computer Science Principles. These approaches target inclusive access to computer science by highly scaffolding the introduction to difficult CS constructs (e.g., conditional logic, variables, and control structures) through tools such as visual block-based programming languages.

A popular program created to educate students about computer science through robotics is RoboScape (3). RoboScape is a program that gives users control of robots through block-based coding and is provided as a service, or an add-on, in NetsBlox (4), a block-based programming language (BPL) with networking capabilities. In a sample study of 24 students using RoboScape as the main learning tool, CT skills of high school students increased by 21%, as measured by a pre- and post-survey of CT knowledge. This growth demonstrates the benefits of teaching computer science in coordination with other mediums of learning, such as robotics. Alongside the growth in CT skills, students regularly arrived at the camp early and hardly paid attention to their mobile devices, expressing increased interest in CS. Although BPLs have shown to be successful in gaining interest, it is imperative for one to also gain exposure to text-based programming languages (TPLs). Early exposure to TPLs can support important software development skills such as debugging syntax errors, a common difficulty for novice programmers (5).

Therefore, the goal of this project was to create a new TPL platform based on the robot controlling features of RoboScape in order to incorporate the advantages of learning TPLs using the strengths of the RoboScape platform. The platform that was created in this project was based on Python, which is widely considered an introductory TPL due to its simplistic syntax and extensive usage in many technology companies such as Netflix, Facebook, and Google (6). The created platform eases the transition from a BPL like NetsBlox into learning TPLs for middle and high school students through the application of skills learned in earlier RoboScape modules. The platform developed in this project has the end goal of teaching computer science to middle and high school students, using robotics as the medium. Because of the general interest from the youth in robotics, it can be hypothesized that a text-based robotics platform would increase one’s computational thinking skills and proficiency in that TPL, as measured through pre- and post-tests. An expected outcome is that the created platform can be used to teach middle and high school students Python, which may serve as the foundation for other, more difficult, TPLs. Additionally, a growth in CT skills can be expected along with the newfound knowledge of TPLs.

MATERIALS AND METHODS.

Programming.

Several different programs and languages were required for the development of platform. The library, or a group of programs that perform a specific task, was developed in Python due to the required implementation of the library in any script that utilizes the robots. Likewise, the server was scripted in Python in order to maintain consistency throughout the platform and properly encode and decode information between different components. The script to be used by all robots was programmed in Propeller C, the proprietary programming language for Parallax. All Propeller C scripting was done in Simple IDE, the integrated development environment used by Propeller C. All scripting done in the development of the platform was done in Microsoft Visual Code Studio, which allowed for easy GitHub interfacing and real-time debugging and syntax-checking.

Platform.

The platform was structured in a hierarchy (Figure 1) that consisted of three components necessary for the successful execution of communication between a user and a robot: the robot itself, the server, and the client/agent. As a whole, a user could prompt a robot to drive,

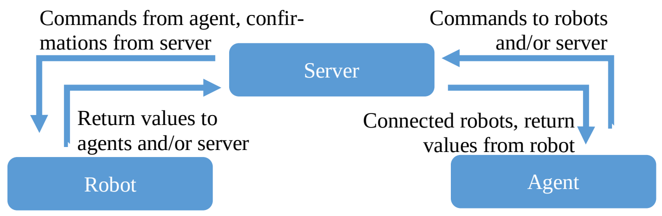
Commands from agent, confirmations from server
Commands to robots and/or server
Server
Return values to agents and/or server
Connected robots, return values from robot
Robot
Agent

Figure 1. A flowchart representation of the platform, starting from the agent to the robot and back.

beep and return the range of an object in front of it or the rotation of its servo motors in ticks.

Agent.

The agent was structured in the form of a library where a user can create an instance of the library's class to allow for seamless robotics control through object-oriented programming. The library was created with a great level of abstraction in mind through the use of simple class methods that do not require one to control a robot with knowledge of what and how string-based commands are being transferred.

The library was built in conjunction with a client script that handled all messages received by and sent from the agent. This client was based on a datagram or user datagram protocol (UDP) socket in which data was sent and read in packets. The socket module from Python was used to create a socket in the script. In a separate thread to allow for simultaneous reading and writing, the client socket read packets continuously from the server. Commands that were read were processed based on a series of conditional statements checking for a specific character that started each command. These characters represented different types of information that was sent, such as the range and servo ticks. In these conditional statements, certain bytes would be converted into integers in order to be returned to the user.

When a library function was used, a command was sent to the client script in order to send data to the server. These commands consisted of a character that corresponded to a certain function that a robot would complete and the MAC address of the specified robot. If a return value was required, the function would hang until the client socket received the corresponding return value from the robot.

Server.

The server controller was structured as an executable Python script that required no user input or manipulation to run successfully. Similar to the agent, the server used the UDP protocol for its server socket script in order to connect with the agents and robots simultaneously and read data coming into the socket continuously in a separate thread. In the controller, data that came through the socket was sent to a given function for processing.

In the specified function, messages were processed based on the starting character and the length of the command. The message was then sent to the receiving component, using its MAC address as an identifier. In the process, the bits that comprised of the message were then rearranged in order for the sending component's identifier to become the identifier of the receiver, allowing for the sender to receive a response. Messages that were sent to the server, such as identification messages or messages that requested a list of connected robots, contained the sending MAC address and the character that differentiated these two functions. Overall, there were five functions that the server processed: setting the speed of the motors, accessing the range of an object, retrieving the rotation value (or ticks) from the motors, causing the robot to beep, and retrieving a list of connected robots.

The server controller had functionality to maintain a list of robots and clients, containing their MAC addresses, IP addresses, and component types, that were connected due to identification messages that were sent from robots or clients upon initialization.

Robot.

The robots that the platform is fully compatible with are Parallax ActivityBot 360˚ Robots (7). These robots contained two servo motors for driving, a buzzer, an ultrasonic sensor for reading the range of a path-impeding object, a WiFi module, and a pair of whiskers that sense touch.

The Propeller C script that each robot was flashed with consisted of a function that continuously read from the server through the WiFi module. Each command was processed by its starting character or an exact comparison with a command. To initialize the robot, the module ran commands to fetch its MAC and IP addresses. The robot then continued to send initialization messages to the server until confirmation was received. Until the robot was powered off, the robot executed the varying library functions that related to its sensors. If the range or tick function was executed, a returning message was constructed with integer to byte conversion and the MAC address of the agent that requested the information.

Pilot Study.

In order to test and measure the functionality, usability, and educational ability of the platform and the curriculum designed, a pilot study approved by the Institutional Review Board was conducted (Figure 2). Five high school students with little to no prior programming experience were recruited from a programming workshop and took part in a pre- and post-test.

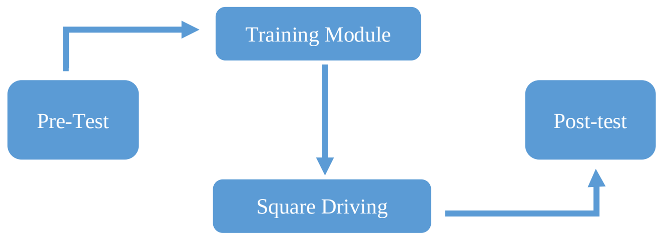

Figure 2. A graphical abstract of the pilot study execution.

Prior to the pilot study, one laptop for each student was equipped with Python 3.7, the library, and the student files and documentation installed. The Open Broadcasting Software (OBS) recording software was also installed in order to record student engagement and participation throughout the study. Paper/pencil pre- and post-tests which comprised of ten multiple choice and open-ended questions were created and mirrored each other. These tests were used to assess knowledge of specific topics, including loops, conditional statements, input and output, boolean expressions, arithmetic operations, lists, string immutability, and class design. Due to these skills being necessary for the robotics square driving task, a training module was also developed for these topics. Documentation for the library and for Python basics, adapted from an online textbook (8), was used as student guides while working through the module and the driving task.

During the pilot study, pre-tests were administered to begin. The participants then engaged in the training module, which consisted of six varying tasks and lasted approximately one and a half hours. Students then participated in the driving task, which consisted of the students programming a robot to drive in a square, that lasted approximately thirty minutes. Post-tests were administered after.

Statistical Analysis.

Statistical analysis was conducted using Microsoft Excel. Averages and sample-based standard deviations were calculated for all of the pre- and post-test scores. To test for a significant difference between the pre- and post-test scores, a two-tailed and paired t-test was conducted, using a p-value of $p<0.05$.

RESULTS.

During the pre-and post-tests, scores were collected in order to determine whether the pilot study led to greater knowledge in computational thinking skills.

The average score increased from the pre-test to the post-test. However, the t-test resulted in a p-value of 0.19, indicating that there was not a significant difference between the pre- and post-test. The stand

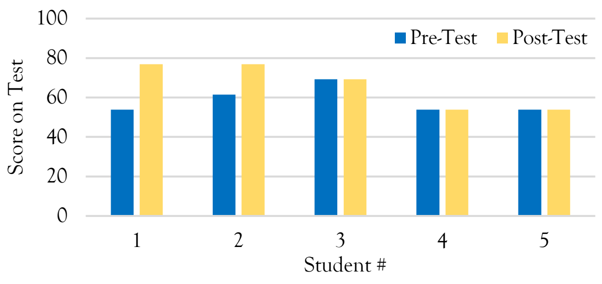

Figure 3. Pre- and post-test scores for each student, in percent accuracy.

ard deviations (Figure 3) for the two tests, 6.88% and 11.67% respectively, were higher than expected, which could have been due to the very small sample size of the study.

The most growth from pre- to post-test was seen in the questions that tested for list indexing, Python syntax, and arithmetic operations. Questions that tested computational skills excluding Python were answered correctly the most often. However, in both tests, all students missed questions that tested class design, string immutability, and import statements.

DISCUSSION.

This project sought to develop a platform that could be used to teach students Python in the context of robotics. Before the pilot study, the platform was projected to affect students' knowledge of Python and CT skills, through a growth of scores from the pre- to post-tests. Based on the findings, there was a slight increase in scores from the pre-test to the post-test. Additionally, the topics addressed in the study, specifically list indexing, Python syntax, and arithmetic operations, were the same topics that experienced the most growth. These results suggest that the study, if prolonged, might have a profound effect on the learning from the pre- to post-tests. The lack of significant learning gains revealed in the pilot study implementation results may be due to the short implementation time, and therefore, further time is needed to instruct the curriculum before a full assessment can be made.

Although the platform was carefully planned, the successful execution was difficult to implement. One issue that arose in the development of the platform was that the built-in method that converted integers to bytes would not accept signed integers, meaning that negative inputs would crash the system, as in the case of turning servos. In order to mitigate the issue, another built-in method from a different package, struct, was used, which accepted signed integers. Also prevalent during the development phase was that bytes could not be shared over different processes due to the process' inability to share memory, not allowing for different methods that read and accept connections indefinitely to share messages. The solution that was found was to remove all usages of processes and switch them to threads. Although threads could have memory leakage issues, potentially crashing the system, messages were able to be shared through different methods. On the robot side in the platform, one initialization message that included metadata such as the robot's MAC and IP addresses could not be picked up by the server, even while running the server method that receives data continuously. Therefore, it was seen that the initialization message from the robot would need to be sent continuously until a confirmation message from the server was received.

One potential improvement for the efficacy of the platform is to implement a simple UI with logging functionality that allows to relieve confusion of constituents in the platform as well as to allow for more data for potential observations of the study. Another potential improvement is to add functionality for pinging all robots which would also relieve any confusions that could be seen by messages being lost in processing. Likewise, implementation of exception handling in the platform would be one room for improvement because one simple issue would not crash the entire system. On the development side, implementing external IP addresses when hosting and receiving would allow for the constituents on the platform to be on different network connections. This would simplify setup procedures that would be done in the start of a new study on the curriculum side of the project. Given known difficulties in introductions to TPLs, increasing training time and tasks in Python may aid in increased abilities in the target TPL, and in particular, applications of class design, string immutability, and import statements. Lastly, in order to supplement the cybersecurity program done in Roboscape, a cybersecurity curriculum and its accompanying functions in the platform should be implemented to also introduce students into cybersecurity in TPLs.

CONCLUSION.

The platform developed can be used for education of Python and computer science skills. Additionally, the curriculum that was developed alongside the platform covers introductory topics needed to introduce students into Python and for Python courses. Even when under strict time constrains and facing technical issues, students demonstrated some knowledge gains as indicated by their pre-post-test results during the implementation of the study. This suggests that with a more generous allotment of time, a significant difference could be achieved. In summation, the curriculum and platform combined have the potential to induce growth of computational thinking skills.

ACKNOWLEDGMENTS.

I would like to thank Dr. Deweese for advising this project and assisting in the creation of research outputs and Bernard Yett for mentoring me throughout this project and assisting with procuring materials. I would also like to thank Gordon Stein for assisting me with the robots and Propeller C debugging. Nicole Hutchins assisted me with the creation of the pilot study and curriculum materials. I would like to thank Dr. Biswas for providing the basis of my research through his leadership of the lab. I would also like to thank fellow classmate Mert Sekmen for assisting me with statistical analysis.

REFERENCES

1. Software Developers. Bls.gov (2019), (available at https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm#tab-6).
2. 2015 college computer science graduates v. open computing jobs. Atlas (2018), (available at https://theatlas.com/charts/B1rFqVkjl).
3. Á. Lédeczi et al., Teaching Cybersecurity with Networked Robots. Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19 (2019).
4. B. Broll et al., Journal of Parallel and Distributed Computing, in press.
5. S. Grover, S. Basu, Measuring Student Learning in Introductory Block-Based Programming. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education - SIGCSE '17 (2017).
6. M. Makai, Companies using Python. Fullstackpython.com (2020), (available at https://www.fullstackpython.com/companies-using-python.html).
7. ActivityBot 360° Robot Kit. Parallax.com (2020), (available at https://www.parallax.com/product/32600).
8. A. Johansen, Python: the ultimate beginners guide! (CreateSpace Independent Publishing Platform, ed. 2, 2016).

Dawit Girma is a student at Martin Luther King Jr Academic High School in Nashville, TN; he participated in the School for Science and Math at Vanderbilt University.