Teaching Cybersecurity with Networked Robots

Ákos Lédeczi, Miklós Maróti, Hamid Zare, Bernard Yett, Nicole Hutchins, Brian Broll, Péter Völgyesi, Michael B. Smith, Timothy Darrah, Mary Metelko, Xenofon Koutsoukos, Gautam Biswas
Vanderbilt University
Nashville, TN, USA
akos.ledeczi@vanderbilt.edu

ABSTRACT

The paper presents RoboScape, a collaborative, networked robotics environment that makes key ideas in computer science accessible to groups of learners in informal learning spaces and K-12 classrooms. RoboScape is built on top of NetsBlox, an open-source, networked, visual programming environment based on Snap! that is specifically designed to introduce students to distributed computation and computer networking. RoboScape provides a twist on the state of the art of robotics learning platforms. First, a user's program controlling the robot runs in the browser and not on the robot. There is no need to download the program to the robot and hence, development and debugging become much easier. Second, the wireless communication between a student's program and the robot can be overheard by the programs of the other students. This makes cybersecurity an immediate need that students realize and can work to address. We have designed and delivered a cybersecurity summer camp to 24 students in grades between 7 and 12. The paper summarizes the technology behind RoboScape, the hands-on curriculum of the camp and the lessons learned.

CCS CONCEPTS

• **Applied computing** → **Interactive learning environments**; *Collaborative learning*; • **Computing methodologies** → Distributed programming languages.

KEYWORDS

visual programming, robotics, cybersecurity, computer science education, Snap!, NetsBlox

ACM Reference Format:
Ákos Lédeczi, Miklós Maróti, Hamid Zare, Bernard Yett, Nicole Hutchins, Brian Broll,, Péter Völgyesi, Michael B. Smith, Timothy Darrah, Mary Metelko, Xenofon Koutsoukos,, Gautam Biswas, Vanderbilt University, Nashville, TN, USA . 2019. Teaching Cybersecurity with Networked Robots. In *SIGCSE '19: 50th ACM Technical Symposium on Computer Science Education, February 27–March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287450

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

ACM ISBN 978-1-4503-5890-3/19/2…$15.00
https://doi.org/10.1145/3287324.3287450

1 INTRODUCTION

Educational robotics is widely used to introduce K-12 students to computer science and engineering [1, 8, 11]. Building and controlling robots have broad appeal for young learners. The typical means of programming robots is writing the program in either a block-based or a simple text-based language, connecting the robot to the computer via USB, and then downloading the program to the robot. In turn, the program executes on the microcontroller of the robot. While there is nothing wrong with this approach, this paper presents an alternative that has certain advantages.

RoboScape relies on WiFi-enabled robots. Each robot has a fixed program that accepts commands transmitted wirelessly and can send back sensor readings over WiFi as well. The robot program that a user of RoboScape is expected to write is, in essence, a smart remote controller. The program runs locally in the browser, enabling a rapid development cycle and making debugging easier. It also makes it possible to control multiple robots from the same program that, in turn, can carry out a task in a collaborative manner.

The programming interface of RoboScape is implemented as a NetsBlox service. NetsBlox is an open-source, networked, visual programming environment based on Snap! [15] that is specifically designed to introduce students to distributed computation and computer networking [2, 3, 12]. RoboScape introduces another unique feature: wireless commands from the users' programs and sensor messages from the robots can be overheard by the other users' programs in the room. Hence, students can eavesdrop on and even take control of others' robots. This provides an excellent opportunity to illustrate the need for cybersecurity and to teach it in a hands-on manner.

The rest of the paper is organized as follows. First we present a brief overview of NetsBlox and then describe the technical details of RoboScape. Section 4 outlines curricular opportunities Roboscape enables that we have used to teach cybersecurity to K-12 students during a week-long summer camp. We conclude by presenting the lessons learned in delivering the material to 24 students and a brief summary of related work.

2 NETSBLOX OVERVIEW

NetsBlox adds a few carefully selected abstractions to Snap! that enable users to create distributed applications. Remote Procedure Calls (RPC) provide access to a set of online data sources such as maps, weather, seismic information, astronomy imagery, etc. and provide useful utilities such as plotting or cloud variables. An RPC is just like a function, i.e., a custom block in Snap!, but it runs remotely on the NetsBlox server. Note that an RPC is more than just a call to a web API. Related RPCs are grouped together into services that

can have state information, can cache data, and optionally can send messages to the caller. For example, the map service returns an image from Google Maps but maintains information for subsequent coordinate transformations from screen coordinates to latitude and longitude and vice versa. It also caches maps to avoid unnecessary API calls.

Figure 1 shows a sample RPC calling block. The first pull down menu lets the user select the service. Picking a service automatically populates the second menu with the available RPCs of the given service. Once an RPC is selected, the block re-configures to show the required input arguments.

Figure 1: A sample RPC call block

NetsBlox also incorporates message passing. Messages are similar to events already present in Snap!, but they also contain data fields and can be sent across the network to other NetsBlox programs running on different computers. A message has a type associated with it specifying its name and data fields. A pair of corresponding send and receive blocks are shown below. Notice the last field of the send block specifying the address the message is to be sent to. There are a number of ways NetsBlox supports message addressing that is beyond the scope of this paper. The interested reader is referred to [3]. Note that NetsBlox simulates peer to peer messaging, but in reality every message goes through the NetsBlox server.

Figure 2: Message sending and receiving blocks

RPCs and messages enable the creation of a wide range of distributed applications and consequently, they open the door to teaching some of the fundamental concepts of distributed computing.

3 ROBOSCAPE

RoboScape is implemented as a NetsBlox service. Robot commands are RPCs. The robot's responses are messages. There are two versions of RoboScape. The first one defines a separate RPC for each different type of command, e.g., *set speed*, *get range*, etc. It also uses different message types for different sensors. The second one was designed specifically for cybersecurity education. It has a generic *send* RPC with two fields: robot and command. The command is a text field where the user needs to provide the command and its arguments as text, e.g., *"set speed 50 50"*. There are only two message types: *robot command* and *robot message*. The *robot command* is for eavesdropping on other users' commands. The NetsBlox server sends this message to every client who has registered to listen on the given robot. Similarly, the *robot message* is sent by the server to each registered client upon receiving a message from a robot. The payload of these messages is also text. Making the communication text-based makes it easy to motivate and implement encryption. The rest of the paper covers this text based communication version of RoboScape only.

The architecture of RoboScape is shown in Figure 3. All clients running in a web browser are connected to the NetsBlox server via the internet. Similarly, the WiFi enabled robots are connected to the NetsBlox server as well. In other words, all communication goes through the server. The communication between the clients and the server is the standard NetsBlox method: http requests for RPCs and WebSockets for messaging. The robots and the server use UDP for communication. The robots run a custom C program that receives the commands, executes them, and sends messages back to the server. The NetsBlox server code was not modified. RoboScape was added as an additional service akin to any other service NetsBlox supports.

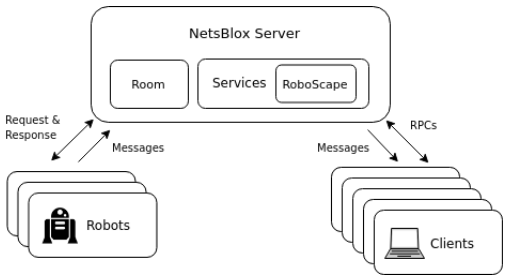

Figure 3: RoboScape architecture

The RoboScape prototype has been implemented with Parallax ActivityBot 360 robots. The cost of the robot is $250. The platform has two main wheels driven by motors with optical encoders, an ultrasonic range sensor, a couple of touch sensors ("whiskers"), a buzzer, a custom button, two LEDs, and an XBee WiFi module. See Figure 4.

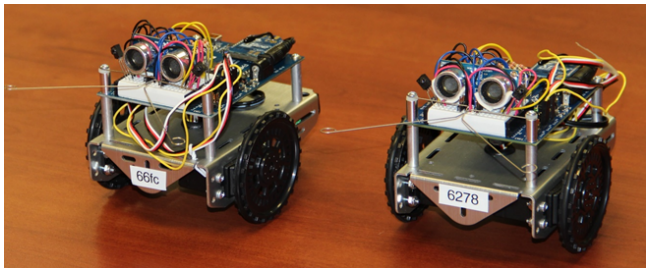

Figure 4: Parallax ActivityBot 360

The basic commands are shown in Table 1. The commands can be overheard as *robot command* messages. Furthermore, when the robot receives the given command, it replies with an acknowledgement containing a time stamp. These, in turn, are sent to the registered clients as *robot messages*. So, if a user registers to receive communication from a given robot and it calls the *send* RPC, it will receive a *robot command* message from the server (simulating overhearing a command) and a *robot message* from the server simulating overhearing the robot's acknowledgement. If the command

Table 1: Basic RoboScape commands

Command	Arguments
is alive	RID (Robot ID)
set speed	RID, left, right
beep	RID, duration, pitch
get range	RID
get ticks	RID
set led	RID, A/B, on/off/toggle
listen	RID

requested a sensor reading, e.g., "*get range*", a third message will be sent with the actual sensor values. The following *robot messages* are supported:

Table 2: Robot messages

Message	Fields
alive	RID (Robot ID), time
beep	RID, time, duration, pitch
speed	RID, time, left, right
range	RID, time, range
ticks	RID, time, left, right
whiskers	RID, time, left, right
button	RID, time, isPressed

To illustrate how easy it is to use RoboScape, consider the simple but fully functional project shown in Figure 5. The program enables the user to drive the robot using the arrow keys on the keyboard. When the green flag is clicked (the standard way to start a program in Scratch, Snap! and NetsBlox), we call the listen RPC of the RoboScape service that registers the client with the server that it wants to listen to the specified robot. Note that the robot ID used is the last few digits of its MAC address.

The arrow key handlers simply use the *set speed* command to specify the speed of the left and right wheels, respectively. The space key serves as the emergency stop button. Finally, there is the handler of the *robot message* message. We parse the message and if its first word is *whiskers*, we stop the robot. That is, if the robot encounters an obstacle, it stops.

From a technical point of view, an interesting question is how network latency effects the performance of the robot programs. A robot command travels from the web browser of the user to the NetsBlox server running in the cloud and then down to the robot. The acknowledgement and/or any sensor values travel back the same way. From a user experience point of view, this delay is not noticeable. When running a program like the one depicted in Figure 5, hitting a the arrow keys starts the robot within a fraction of a second.

To quantify the latency, we carried out an experiment. We issued a command in a loop as fast as possible. Since send is a blocking call returning either a sensor value (*get range* or *get ticks*) or a true or false whether the command succeeded or not (*set speed* or *beep*), the time between two such commands includes the latency of the full round trip. We tried different commands and two different local WiFi networks: a relatively slow mobile personal hotspot and a

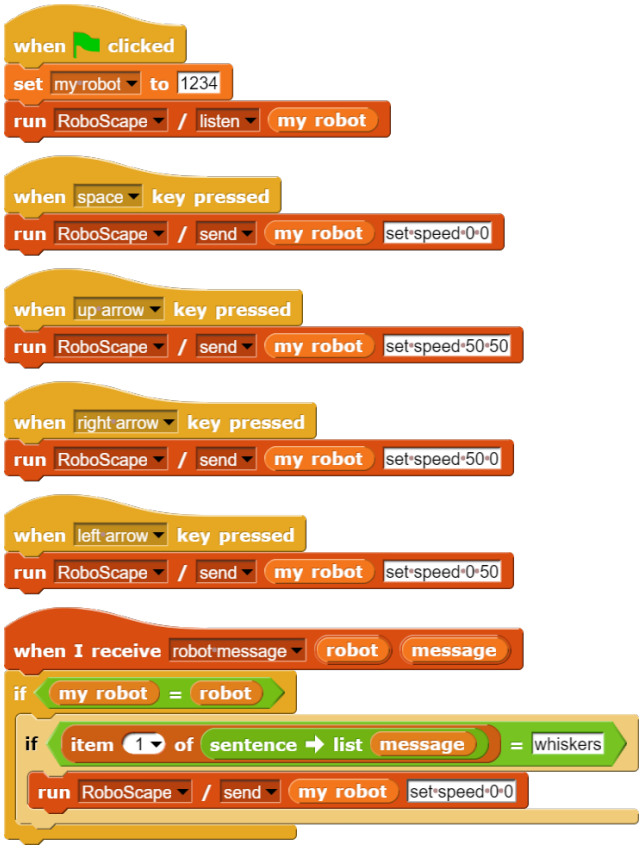

Figure 5: A simple RoboScape project implementing driving the robot with keyboard commands

fast WiFi router. The average latency across all experiments came to between 100 and 120 milliseconds. That is faster than human response time. Interestingly, if we measure the time when a message arrives in response to a command (as opposed to a return value), the delay doubles. Since the network latency should be about the same, this added delay must be due to extra processing on the server and the client dealing with message passing. While this performance is more than satisfactory, the system can be sped up significantly by running a copy of the NetsBlox server on the local network as opposed to the Amazon cloud.

4 CYBERSECURITY

This section briefly summarizes which cybersecurity concepts can be taught using RoboScape.

DDoS. Since all one needs to know is the robot ID in order to issue commands to it, it is really easy to mount a (Distributed) Denial of Service (DDoS) attack. Instead of taking control of a robot overtly by issuing *set speed* commands, we show the students how to issue *set led* or short *beep* commands rapidly in a loop. As more and more students start doing this, the robot starts to lose real messages and its behavior becomes erratic. The first task we assign the students is to modify their driving program to detect if there is an attack occurring. The simplest way is counting the number of commands

issued and comparing it to the number of commands overheard. Then we introduce two new commands. The *set total rate* command limits the number of commands (per second) the robot accepts to a specified number. This is of course not effective since the real commands and the attackers' commands are all grouped together. The second command is called *set client rate* and it specifies the maximum rate commands accepted from the same client as well as a penalty period when the robot does not accept any command from the given client whenever the limit is crossed.

The challenge problem for the students is a race we call tug of war. Two students control the same robot. The robot is placed in the middle of the room and the objective for each student is to drive the robot to his or her side of the room. The client rate is set to an agreed upon fixed number. The students can use manual driving, that is, they can use the keyboard of their laptops for various commands. The challenge comes from the fact that you want to issue the last command before the rate limit is reached, because afterwards you cannot issue any new commands for the duration left of the current second.

Encryption. Students quickly realize that it would be much better if the robot did not accept commands from anybody else. This presents an opportunity to introduce the concept of encryption. The robots support a simple Caesar's cipher. The corresponding command is called *set key* and it sends the shift amount to the robot. Once the robot receives a new key (the default is 0 at startup), it decrypts every message using the key before sending it and it only accepts commands encrypted with the very same key.

There are two obvious weaknesses with this scheme. Some students figure out the first one quickly: since there are only a limited number of commands and, for example, if you want to drive your robot, you must use *set speed*, one can relatively easily break the encryption by brute force even using a relatively slow, blocks-based language. The second weakness is that the very first *set key* command is not encrypted.

Figure 6 shows the code that listens to *set key* commands sent to a robot and steals the key. It uses the last stolen key to decrypt the command and extract the new key. Since the initial key is zero, if the attacker intercepts the very first *set key* command, the program will be able to eavesdrop on the robot indefinitely, no matter how many times the key is changed. This simple example introduces the students to the need of secure key exchange.

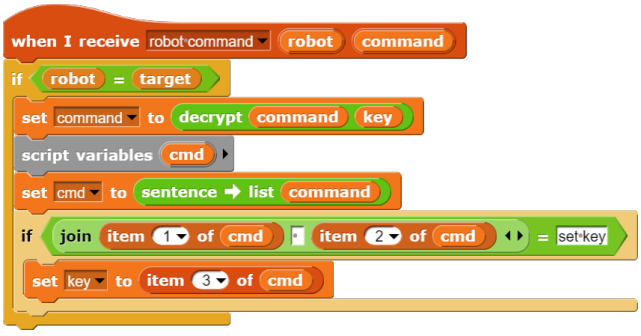

Figure 6: Eavesdropping on a robot and stealing the key when a *set key* command is overheard

Hardware key/secure key exchange. A simple way for secure key exchange in RoboScape is to have the robot generate the key and never send it across the wireless network. When the user pushes the button on the robot for two seconds, the robot generates a new key and "plays it" on two LEDs. There is one LED for bit 0 and another for 1. The robot blinks one or the other for half a second from the most significant bit to the least. The student writes this number down, converts it from binary to decimal, and types it in their program. Once they do this, the key thief program of Figure 6 will no longer work.

The brute force code breaking can also be made much harder: instead of a single shift value for the Caesar's cipher, RoboScape supports using a list of shift values. The encryption then proceeds by shifting the first character of the message by the first shift value, the second by the second and so on in a circular fashion. The robot itself generates four 4-bit shift values. Knowing this, brute forcing a *set speed* command may still be possible (even though the number of possible combinations are over 64,000 as opposed to fewer than 100), but if the first thing the user's program does is change the key to a new list of more than 4 values, and then continue to change it frequently, the attackers will not be able to keep up.

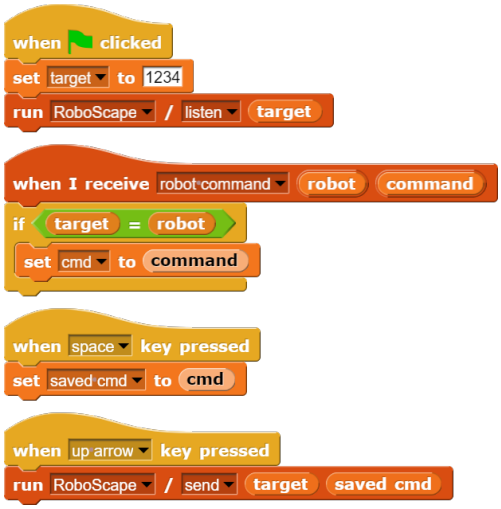

Figure 7: Capturing encrypted commands and replaying them on demand

Replay attacks. At this point of the curriculum, students should have secured their robots via the aforementioned, scaffolded tasks. Instruction on additional robot interference then takes form in the application of replay attacks. Even though the attackers cannot break the encryption, they can still overhear messages and reissue them themselves. The program in Figure 7 allows the attacker to capture any commands the user sends to the robot and even though they cannot decrypt it, the person can observe the robot's behavior, and issue the saved command at a later time. For example, they can wait until the robot turns, press the space button to save the last command overheard (presumably the turn command), and then they can cause the robot to turn any time they choose to by pressing the up arrow.

There is a simple fix to this problem with RoboScape: each command can be prepended by a sequence number. If a robot receives a command that starts with a number, it will only allow commands that also start with a number from then on. Furthermore, it will only accept numbers that are strictly greater than the last received sequence number. To allow for lost messages, it accepts numbers in the range $[seq + 1, seq + 100]$ where seq is the last received valid sequence number.

Table 3 lists the cybersecurity related commands.

Table 3: Cybersecurity specific commands

Command	Arguments
set key	RID (Robot ID), key
set total rate	RID, rate
set client rate	RID, rate, penalty
reset rates	RID
reset seq	RID

5 LESSONS LEARNED

We conducted two one-week long camps with a total of 24 students from grades 7-12. The students were split approximately evenly between male and female. A high school teacher also participated in one of the camps. Based on the curricular affordances provided by our environment, we completed a domain unpacking of key CS concepts and practices available in the K-12 CS Framework [7] to support the design of a scaffolded introduction to cybersecurity and a prepost assessment to evaluate learning gains. Key concepts include computational thinking (CT), algorithms and programming, networks and the internet, and the impact of computing, with a focus on the practice of creating computational artifacts. We could not assume prior computer programming experience, so we started with an introduction to NetsBlox programming that included increasingly difficult tasks designed to target the CT and programming concepts of control structures, variables, and modularity. The second day was dedicated to learning RoboScape by creating simple robot driving programs. This day not only allowed for additional programming practice, but it also provided for a preliminary introduction to key networking concepts to support the more advanced cybersecurity tasks to come. The final three days targeted our cybersecurity-specific material through scaffolded tasks introducing DDoS, encryption, and other concepts. We additionally emphasized the framework concept of impacts of computing through cybersecurity guest lectures applying curriculum content to real world applications. The hands-on portion of the curriculum is summarized in Table 4.

Our prepost assessment was designed based on the concepts and practices implemented in our curriculum and was sectioned into CT, networking, and cybersecurity. For instance, given the assumption of little to no prior programming experience and the introduction to control structures on Day 1 of the camp, the CT section of our assessment included a question on the use of conditional statements to better assess prior knowledge and to evaluate student understanding post-camp. In addition to learning gains, we implemented a prepost survey targeting students' self efficacy in programming, attitudes towards technology, and task values placed upon robotics and cybersecurity to evaluate student attitudes toward the curriculum content and their abilities.

Table 4: Daily schedule

Day	Topics
1	Intro to NetsBlox programming
2	Intro to RoboScape programming
3	Attack detection, DDoS, its mitigation
4	Encryption, secure key exchange
5	Replay attack, its mitigation, final project

Table 5 reflects our results, showing that most categories improved significantly (p-score < 0.05) from pre-test to post-test. This is reflected in the average scores from the assessment (on a 0-1 scale) and survey (on a 1-6 scale). Also included is the effect size experienced in each area; effect size is simply the average of post-test scores minus the average of pre-test scores, which is then divided by the standard deviation of the full sample. As shown here, students experienced remarkably significant improvements in all sections of the questionnaire; they were specifically addressed during the camp in the form of robotics applications and throughout several guest lectures, so it is reassuring to see the results. Additionally, students expressed improvement in two sections of the motivational survey, showing that they feel more comfortable in completing programming tasks related to robotics and cybersecurity while also appreciating the value of technology. This makes sense considering the topics of the camp as well as its elective nature. In addition, students could be heard saying the following during post camp interviews that nicely supports our results: "The robots really helped because we got to apply what we've learned to an actual situation where we can see the results." "It's interesting and I really like coding now... before I kind of viewed it [coding] as something that could spark my interest, but now it [the camp] has heightened my love for coding and I want to explore more."

Table 5: Survey Results

Category	Pre Avg.	Post Avg.	P-Score	Eff. Size
CT	0.70	0.91	0.001	0.833
Networking	0.58	0.80	0.001	0.845
Cybersecurity	0.61	0.81	0.002	0.674
Self-Efficacy	4.81	5.42	<0.001	0.627
Tech Att.	5.39	5.65	0.011	0.368
Task Values	5.27	5.55	0.176	0.159

We would next like to briefly address the result which did not experience significant improvement. While there was some amount of improvement in the value students place on robotics, cybersecurity, and programming tasks, it was not enough to be significant. The main issue here seems to be that students already valued those areas highly, so there was not as much room for improvement as in the other areas.

There is plenty of anecdotal evidence of the success of the camps as well. Student engagement in both camps was very high. After the second day, that is, after the students started programming the

robots, they regularly arrived half an hourly early and took very short lunch breaks. One could hardly see any mobile phones even though we did not ban them. None of the students went on social media or played online games during the camp. The participating high school teacher remarked how unusual this was with teenagers, saying "I did not see them on cell phones, they were engaged with programming their robot."

The anonymous survey at the end of the camp also showed very positive attitudes toward the camp. Most students expressed how much fun they had and how much they learned. The few negative comments were from students who had no previous programming experience (and were probably the younger learners) who had a somewhat harder time keeping up with the fast-paced camp. To illustrate students' attitudes about the camp, here are some answers to the question, "What was your favorite part of the camp?": "Writing brute-force programs for 4-key encryption." "...being able to apply new and exciting features to the robots that I never thought I would be able to do. It really makes we want to get more engaged in computer science as well as networking." "...hacking people's robots and watching them become confused and making them think their robots are broken." "... racing other teams' robots and fighting with others to get the robot to go a certain direction."

Several students identified encryption as the most difficult concept they learned. Others listed various aspects: "Everything I tried that didn't work made sense once I broke it down and really thought about it except for maybe the command to stop and turn around the robot when its whiskers hit an obstacle." "For a day or two I couldn't wrap my head around lists for some reason but it clicked. Everything else was also a challenge but I eventually got the hang of most of the content!" "The geometry killed me, and some of the robot programming was hard..." The students' engagement is nicely illustrated by this comment: "Lots of complex subjects were taught in the GPS presentation that I studied more when I got home, but it did take me a while to try and fully understand some of the concepts taught."

6 RELATED WORK

The most similar camp to the one we put on seems to be Cyber Discovery and its offshoot, Junior Cyber Discovery (JCD), which covered the same general topic areas of robotics and cyberspace and used a Parallax Boe-Bot for hands-on activities to teach the students about code security, wireless signal transmission, and programming [16]. The use of the Boe-Bot was very prominent during the one reported year of JCD, though those students also designed and build the robots [5]. Another series of camps that has seen rapid growth is GenCyber [13],[10]. One example occurred at Purdue University Northwest [6], where cybersecurity ideas were integrated with game-based learning, using a pair of 3D VR games to teach social engineering, information security, and secure online behavior. Another example built around the GenCyber model was CyberPDX [4]. The GenCyber programming section involved two tools for teaching new programmers - Blockly, which is a graphical, block-based language similar to NetsBlox, and Turtle Graphics in Python. CyberPDX students move on to writing their own encryption program (given a decryption one) or to creating a brute-force attack program. Both camps demonstrated significant learning gains.

Our camp mainly differs from these examples in that attendees were able to focus entirely on how to best program the robots. On the cyber side, our camp was focused on cybersecurity instead of general cyber ideas, as students were tasked with defending their robots while attempting to hack the robots of other groups. These examples also tend to be aimed at different age or experience groups instead of using robotics as a teaching tool for programming and cybersecurity aimed at predominantly high school students.

The Cozmo robot from Anki [14] is a sophisticated yet inexpensive platform. It is being used in introductory robotics course at Georgia Tech (CS 3630), and supports remote control through an SDK. However, it requires a direct Bluetooth connection to a mobile device, and only one robot per device is allowed. As such, it would not be an ideal platform for our purposes. Finally, Sphero is a small robotic ball that is suitable for a robotics and mobile computing course [9]. Students can learn and use simple programming tools on a smart-phone app to send commands or other information to the robots. It is also set up to allow for growth in programming knowledge, from simple block languages up to JavaScript. We plan to provide support for additional robot platforms in the future.

7 CONCLUSIONS AND FUTURE WORK

While both RoboScape and the curriculum of the camp are first prototypes, and the participants were self selected and not representative of the general K-12 student population, we can draw some preliminary conclusions. The apparent success of the summer camps shows the promise of teaching robotics and cybersecurity together in a hands-on manner. The fact that students with no previous programming experience were able to mount cyber attacks and implement simple cyber defense techniques by the end of the week seems to validate the value of the RoboScape approach. Implementing and running the code in the browser made understanding the concepts and designing and debugging the robot control programs easier. One lesson for the future is that we need to create two versions of the curriculum: one with students who have not programmed before and a more advanced version for students who have. Next year, we will develop and deliver two such variants. We also plan to develop a PD week for interested teachers so that they can take the material and implement it in their schools as an after school club. Finally, we'll evaluate additional robotic platforms with different sensors and actuators for potential support by RoboScape and the cybersecurity curriculum.

8 ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Security Agency Science of Security Lablet, the Air Force Research Laboratory under Award FA 8750-14-2-0180 and the National Science Foundation under grants CNS-1644848, CNS-1238959, CNS-1739328, and DRL-1640199. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the US Government.

REFERENCES

[1] F. B. V. Benitti. Exploring the educational potential of robotics in schools: A systematic review. *Computers & Education*, 58(3):978–988, 2012.

[2] B. Broll, A. Lédeczi, P. Volgyesi, J. Sallai, M. Maroti, A. Carrillo, S. L. Weeden-Wright, C. Vanags, J. D. Swartz, and M. Lu. A visual programming environment for learning distributed programming. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 81–86. ACM, 2017.

[3] B. Broll, Á. Lédeczi, H. Zare, D. N. Do, J. Sallai, P. Völgyesi, M. Maróti, L. Brown, and C. Vanags. A visual programming environment for introducing distributed computing to secondary education. *Journal of Parallel and Distributed Computing*, 118:189–200, 2018.

[4] W. chang Feng, R. Liebman, L. Delcambre, M. Lupro, T. Sheard, S. Britell, and G. Recktenwald. Cyberpdx: A camp for broadening participation in cybersecurity. In *2017 USENIX Workshop on Advances in Security Education (ASE 17)*, Vancouver, BC, 2017. USENIX Association.

[5] J. Holcomb, M. Nelson, H. Tims, G. Cazes, and G. Turner. Junior cyber discovery: Creating a vertically integrated middle school cyber camp. In *Proceedings of the American Society for Engineering Education*, June 2012.

[6] G. Jin, M. Tu, T.-H. Kim, J. Heffron, and J. White. Game based cybersecurity training for high school students. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 68–73, New York, NY, USA, 2018. ACM.

[7] K12 Computer Science Framework. http://www.k12cs.org, 2016. Cited 2018 August 31.

[8] F. Klassner and S. D. Anderson. Lego mindstorms: Not just for k-12 anymore. *IEEE Robotics & Automation Magazine*, 10(2):12–18, 2003.

[9] S. Kurkovsky. Mobile computing and robotics in one course: Why not? In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 64–69, New York, NY, USA, 2013. ACM.

[10] T. Ladabouche and S. Lafountain. Gencyber: Inspiring the next generation of cyber stars. *IEEE Security & Privacy*, 14(5):84–86, 2016.

[11] M. J. Mataric. Robotics education for all ages. In *Proc. AAAI Spring Symposium on Accessible, Hands-on AI and Robotics Education*, 2004.

[12] NetsBlox website. https://netsblox.org. Cited 2018 August 31.

[13] B. R. Payne, T. Abegaz, and K. Antonia. Planning and implementing a successful nsa-nsf gencyber summer cyber academy. *Journal of Cybersecurity Education, Research and Practice*, 2016(2), 2016.

[14] J. Skågeby. "Well-Behaved Robots Rarely Make History": Coactive Technologies and Partner Relations. *Design and Culture*, 10(2):187–207, 2018.

[15] Snap!: a visual, drag-and-drop programming language. http://snap.berkeley.edu/snapsource/snap.html. Cited 2018 August 31.

[16] H. Tims, G. Turner, C. Duncan, and B. Etheridge. Work in progress - cyber discovery camp - integrated approach to cyber studies. In *2009 39th IEEE Frontiers in Education Conference*, pages 1–2, Oct 2009.