Paper ID #37045

Work-in-Progress: Bridging the Gap Between MATLAB and Python via ROS to Build Skills in an Introductory Programming Course

Joshua Fagan

Joshua Fagan received a BS in computer science and mathematics from The University of Richmond and a MS in computer science from The University of Tennessee, Knoxville (UTK). He is in his final year of a PhD in computer science at UTK, and will be starting a Lecturer position at UTK in Fall 2022. Joshua is interested in developing and teaching solutions involving robotics, machine learning, and data science.

Amy Biegalski

Amy Biegalski is a graduate of The Ohio State University (BS) and Case School of Engineering (MS, PhD). Among other courses, she teaches Computer Solutions of Engineering Problems in the Engineering Fundamentals Program at the University of Tennessee. She is interested in active and project based learning, and technology based introductory engineering classes.

Work-in-Progress: Bridging the Gap Between MATLAB and Python via ROS to Build Skills in an Introductory Programming Course

Abstract

Robots are prevalent in introductory engineering courses to facilitate kinesthetic learning. This paper describes a new open-source robotics toolbox and its implementation in an introductory MATLAB programming course for engineers. The toolbox was designed to allow students to easily and intuitively program small, low-cost, customizable mobile robots using MATLAB. The MATLAB algorithms are converted to Python commands via the MATLAB ROS Toolbox. We describe the motivation for selecting the software and robotics platform, examples of the labs and projects implementing the robots, the framework of the initial version of the toolbox used in the course, challenges encountered, and the resulting toolbox developed after receiving data and feedback from large scale implementation.

1 Introduction

Robots have emerged as a highly popular educational tool to increase engagement and address the needs of kinesthetic learners [1, 2]. Perceived benefits from robot integration in introductory programming courses include an increase in programming skills, peer learning, and student motivation [3]. It has been observed that along with fostering creativity, using robotics in these courses increases student success [4, 5]. In general, introductory engineering courses have incorporated robots with a goal to increase problem solving skills [6] and overall program retention [7]. However, robots are often a source of frustration to students. McGill observed that to gain benefits in student motivation in an introductory programming course, hardware and software implementations need to be better investigated and developed to sufficiently reduce the frustrations of a novice programmer [8]. Thus our goal was to create a new toolbox consisting of simple commands and house it as a GitHub repository allowing for continual collaborative development to increase usability while reducing the anxiety caused by implementing the software and interacting with real-world conditions.

From our experience, we have found that robots serve as an excellent teaching tool to make programming accessible and are engaging to students of diverse backgrounds. The hands-on, collaborative environment and live feedback from algorithms can make logic and coding more approachable to those more hesitant or timid about their programming abilities. The use of robots can be exciting to students; it can grab the attention of struggling learners. Open-ended problems allow students to express their creativity, and with robust software and hardware expandability, it enables highly motivated students to push their boundaries and develop unique, nontrivial solutions.

Implementation of robotics in this course began in 2015 with iRobot Creates, which were tethered to control computers with a serial cable using a variation of the Esposito toolbox [9, 10]. This toolbox is comprised of features appropriate to a beginning programmer. For example, the serial byte sequence control commands are invisible to the user, instead of drive at a speed, a single line command is used to drive a set distance and stop, and instead of turning via differential wheel speeds, users could specify a turning angle. The following year we added an onboard Raspberry Pi (RPi) for Wi-Fi control and a live camera feed for image processing. With the new platform, the toolbox had to be extensively modified. As we incorporated the robots into more labs and projects, charging the robots on their docking stations became a significant challenge as we could only use a small fraction of the robots at a time. Students expressed frustration with lack of access to charged robots and the robots weren't adequately portable for use outside of the classroom. Other sources of frustration were the inaccuracy of the motor controls and sensors and the lumbering movement. With new funding available in 2020 we sought a lower-cost, more nimble, more portable, smaller robot with battery interchangeability, while maintaining a beginner friendly software platform. We also wanted to more easily accommodate the creative requests of students as they expressed interest in adding peripherals to the robots and multi-robot capabilities.

1.1 In-class Activities

The current class teaches 200-350 total students a semester with 30-60 students in each class session. Students start the semester learning the fundamentals of programming with MATLAB in a purely software environment. After gaining an understanding of the fundamentals, students gain an appreciation for how software can be used to affect physical objects through the robot activities described in this section. In addition to these activities, students continue exploring software specific problems and solutions such as curve fitting and optimization. The presented toolbox is used to facilitate the students engaging in the robot activities throughout the semester.

The course's robotics activities seek to address the course objectives of building programs to solve engineering-related problems, learning to apply common programming practices, manipulating data, and presenting solutions in design projects. Prior to the first robotics lab, students are introduced to basic implementation of functions, conditionals, loops, and fundamental numerical and graphical analysis techniques.

The first robotics lab requires easily attainable goals to provide motivation and confidence for subsequent activities. We provide a short drive and turn function, and have students manipulate it to implement a for loop that will result in a square path traversal. Students utilize their recently learned skills of loop and function writing, but also observe that the square is imperfect. This serves as a good teaching moment for real-world ramifications. The second exercise implements conditionals, where the robot's motion changes based on a sensor reading. The next activity is a competition for student teams to test their skills through development of an autonomous control program. They must either navigate an obstacle course with physical barriers or a "street" with signs and make use of the robot's color identification capabilities.

A second pre-defined lab utilizes MATLAB's image processing capabilities. Students view a live video feed from the robot and are asked to manipulate a snapshot saved as an image file using commands explored in the prelab. Next, they are provided with six example algorithms for their experimentation that rely on MATLAB toolboxes: object recognition, face recognition, text recognition, shape detection and measurement, and advanced image feature comparisons. Student teams select at least two to implement and are asked to make meaningful modifications to the programs. After this two lab series, students begin to observe the parallels between their robots and autonomous car technology.

Equipped with the experiences of basic robot algorithm writing and image processing, students begin open-ended design projects. Projects are centered around a theme, such as an astronaut assistant on Mars. Individuals or pairs of students develop functions to serve as "features" for their assistant. Then, larger teams incorporate the multiple individual pieces into one cohesive project and program. Throughout the design project phase, practical labs continue where students learn additional MATLAB capabilities that can be implemented into their project, including sound processing, custom user interfaces, input validation, and more advanced numerical analysis tools. At the end of the semester, students showcase their projects in personal e-portfolios.

2 Methods

We selected Sphero's RVR robot as the new robotic platform for the course. Our RVR's have an onboard RPi that can be controlled using Python or C++ code. Because the introductory programming course is MATLAB-based, we needed a bridge to allow the students to use MATLAB to control the RPi running Python. For that bridge, we use Robot Operating System (ROS) [11]. Figure 1 gives an overview of how students communicate with and control the robots.

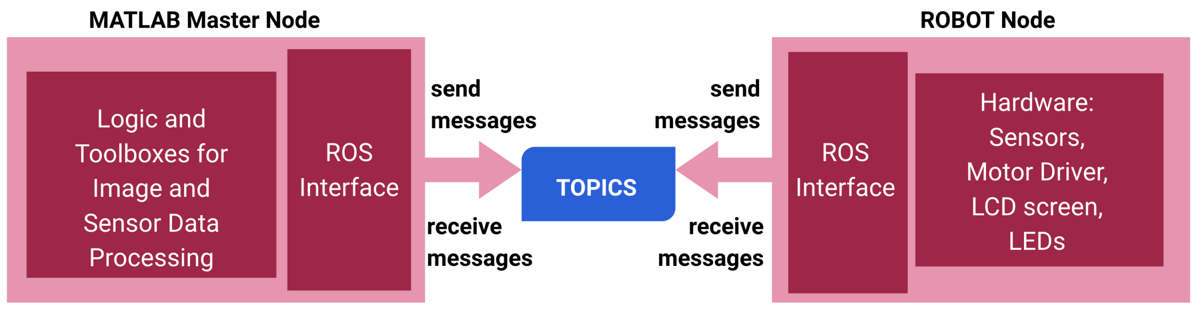

Figure 1: Overview of system design.

2.1 MATLAB

MATLAB has been identified as the language of choice for our students based on its ability to solve and visualize a wide range of problems, its robust and helpful IDE, it is self-contained with an extensive number of built-in toolboxes and functions (e.g, for our labs we use the computer vision, deep learning, and image processing toolboxes), and it offers potential career benefits with its widespread usage in industry and academia. For our beginning programmers, MATLAB offers a low-barrier environment but still translates well as a gateway to other languages. Although we are implementing ROS communication through MATLAB's ROS Toolbox, its use is concealed to the students, they are only working with very basic and descriptively titled commands for control and sensor data acquisition. Based on our experience, this toolbox is of significant value because it allows students to directly or autonomously control the mobile robots through simple MATLAB commands and a more user friendly interface as compared to implementing Python, JavaScript, or the MATLAB ROS toolbox directly with the Sphero Robots.

2.2 Sphero RVR

A small, relatively low-cost robot was essential for our course that serves 200-350 students per semester. We found the Sphero RVR to be satisfactorily nimble for maneuvering through tightly spaced barriers and over obstacles, robust enough to endure overturning and moderate impact, and enticing with a look similar to a smart car. Each robot is stored inside a portable case that allows for project work outside of the classroom hours.

The Sphero RVR has on-board sensors that include a gyroscope, accelerometer, ambient light sensor, and a floor-facing color sensor to determine riding surface RGB color. Like our previous robots, our basic build includes an on-board RPi with a Wi-Fi adapter for wireless control. The RPi is outfitted with a camera for image processing and analysis. The RVR does not have a built-in sensor for obstacle avoidance, so similar to the commercially available SparkFun Kit for the RVR[12], our build includes a time of flight distance sensor and servos for pan/tilt camera capability. To complete the build, as seen in Figure 2, we added a mini-LCD screen for displaying graphics or text, and custom 3D printed mounting plates and rollbars. Our unit cost was $280 but could be scaled down or expanded. With the flexible platform, students can design and 3D print accessories and the multi-channel development board allows opportunity for adding other sensors and components for control and output.

The Sphero API allows for external raw motor control, LED control, sending/receiving IR signals for robot following, and sampling from the 9-axis IMU, accelerometer, and gyroscope. Sphero has made available a Python and node.js SDK for using the RVRs with RPi's and a C++ SDK for Arduinos.

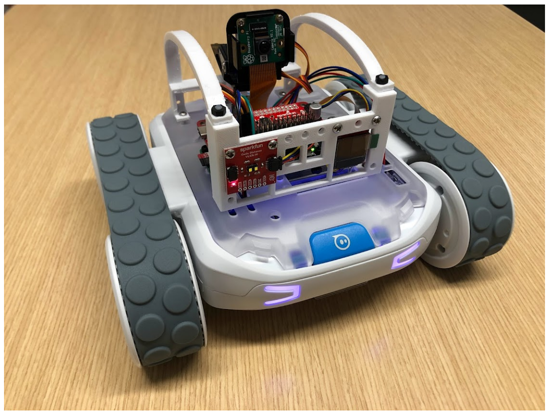

Figure 2: Sphero RVR with custom full build.

2.3 ROS Communication Bridge

ROS is an open source software platform that allows for communication on heterogeneous distributed systems. ROS acts as a bridge between the RVR's RPi and MATLAB running on a single student's computer. Commands can be sent from MATLAB via ROS to the RVR to move the robot and sensor data can be sent from the RVR via ROS to the student's MATLAB for processing and analysis. This toolbox allows students to interact with robots to increase their skills in programming with real world constraints. Further, the toolbox can accommodate any future student-driven sensor expansions.

ROS is structured as a set of nodes that pass messages to each other via topics and services. Each machine can have multiple nodes in use, depending on what processes are running. MATLAB has a built-in ROS toolbox, which allows the developer to create any number of ROS nodes within MATLAB. Similarly, Python has the functionality to develop any number of nodes on the RPi.

2.3.1 Topics

Topics, as shown in Figure 3, are named, typed message streams that facilitate a "many-to-many" communication paradigm. Any node can send data to a topic by creating a ROS publisher that is supplied with the name and datatype of the desired topic. Any node can receive data that is being published to a topic by creating a ROS subscriber that is supplied with the topic's name and datatype. A single topic can have any number of publishers and subscribers linked to it. In this way, any number of nodes can publish information to any number of other nodes in an asynchronous fashion. This method of message sharing is desirable for sharing robot state information or passing data streams from sensors, especially when working with teams combining multiple people and/or robots.

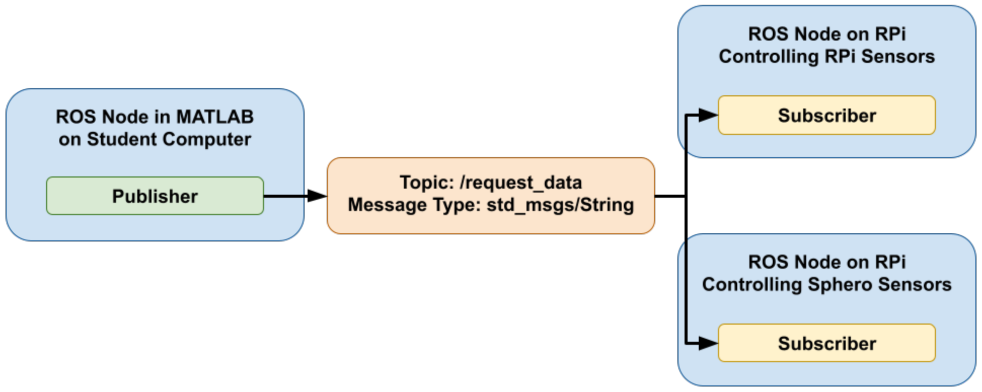
ROS Node in MATLAB on Student Computer
Publisher
Topic: /request_data
Message Type: std_msgs/String
ROS Node on RPi Controlling RPi Sensors
Subscriber
ROS Node on RPi Controlling Sphero Sensors
Subscriber

Figure 3: Visualization of a topic with one publishing node and two subscribing nodes.

2.3.2 Services

Services, as demonstrated in Figure 4, implement a request/response message-passing paradigm. Request and response expectations are created with a set of custom ROS messages, which explicitly set what information is sent to and from a service. A server node hosts a service and a client node can send request messages to the service, whereupon the server node will perform a prescribed action and send back a response message to the client. The client's request message may contain no information other than the desire for a response from the server, or it may contain information pertinent to the request being made. The action performed by the server may be anything (e.g. sampling a sensor, performing a mathematical operation, or driving a motor). An important characteristic of services is that they will block further requests until the prescribed action is performed and a response is sent. This is an ideal form of sharing data from sensors with more control over bandwidth usage than basic topics allow. For example, if the client wants an image from the robot relatively infrequently, they can request an image from a service, the service can snap a picture with the camera, and then the service would send a response to the client with the picture.

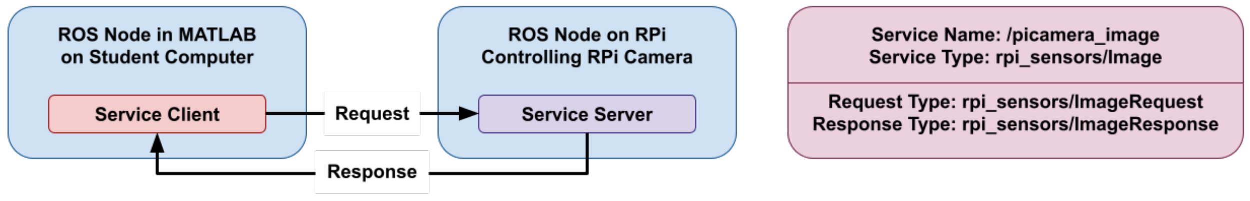
ROS Node in MATLAB on Student Computer
Service Client
Request
Response
ROS Node on RPi Controlling RPi Camera
Service Server
Service Name: /picamera_image
Service Type: rpi_sensors/Image
Request Type: rpi_sensors/ImageRequest
Response Type: rpi_sensors/ImageResponse

Figure 4: Visualization of a service server receiving service client requests and responses.

2.3.3 Actions

ROS actions, visualized in Figure 5, allow for a similar message-passing structure as services, but with some added features that make them ideal for handling messages that result in robot movements. Actions and services are structured similarly as far as there is a server node that hosts an action that accepts messages from a client node. Instead of the client sending a request and getting a response from the server, as in services, actions have three main forms of communication: the action client sends a goal to the action server, the action server sends back periodic feedback on how the action is going, and finally the action server sends back a final result from performing the action. Similar to services, each communication back and forth is done with ROS generated custom messages. For example, if the client wants the robot to drive for five seconds, a goal message may be sent to the server specifying the desired speed and time. The action server would execute the appropriate commands to move the robot motors and may send feedback messages to the action client indicating relevant information such as time spent moving, current location, and current speed. Once the robot has moved for the appropriate duration, a success message may be sent to the action client. One important characteristic that distinguishes actions from services is that actions are non-blocking. This allows for the client to "preempt" goals (i.e. send a goal, which takes over control of the robot before another goal's completion). This allows for functionality, such as allowing the robot to drive indefinitely in one direction, but if it sees an obstacle, it will stop.

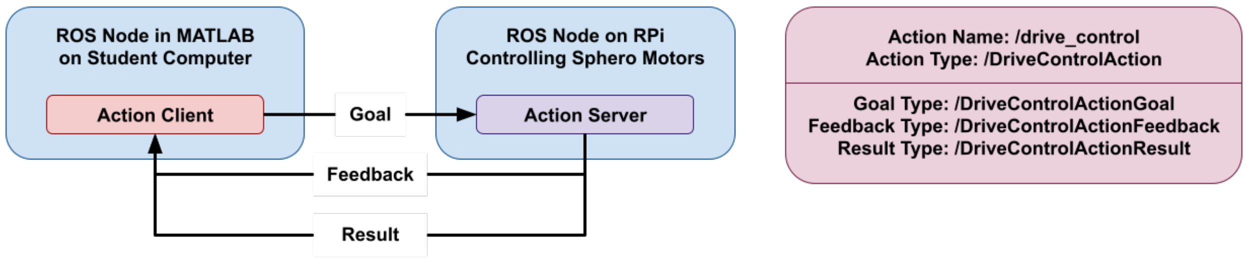
ROS Node in MATLAB on Student Computer
Action Client
Goal
ROS Node on RPi Controlling Sphero Motors
Action Server
Feedback
Result
Action Name: /drive_control
Action Type: /DriveControlAction
Goal Type: /DriveControlActionGoal
Feedback Type: /DriveControlActionFeedback
Result Type: /DriveControlActionResult

Figure 5: Visualization of an action server receiving action client goals and giving responses.

3 Implementation Results

In this toolbox, one node, the master node, is hosted from each student's MATLAB, and each Sphero RVR hosts four nodes: a drive control node, a Sphero sensor control node, a RPi distance sensor control node, and a RPi Camera control node.

The drive control node hosts an action server that uses custom ROS messages to receive goal messages, call Sphero API commands, and send feedback and result messages back to the action client on the MATLAB master node. The Sphero API commands currently called will perform one of the following operations: set wheel speeds independently, turn robot in place for a specified amount, drive the robot backwards at a specified speed for a specified time, reset the heading to be the current facing direction, stop the robot's motion. The goal messages contain data fields that supply the action server with all the information needed to call the appropriate commands with the Sphero RVR API: a string indicating which command to run, two integers indicating left and right wheel speeds, an integer indicating turn angle, an integer for specifying the desired heading, and an integer for specifying drive time. Not all information is pertinent for each command (e.g. time is not used when calling the stop command) and the unneeded data fields are set to zero.

The Sphero sensor control node, RPi distance sensor control node, and RPi Camera control node each stream all of the data the sensors generate in real time to topics that the MATLAB master node subscribes to.

As shown in Table 1, two problems arose with the initial implementation. Firstly, MATLAB changed how custom ROS messages were supported in MATLAB, with the shift from R2020 to R2021. If a node connecting to MATLAB via ROS uses custom messages, such as the drive action server node, MATLAB generates a set of files that lets MATLAB interface and communicate with those custom ROS messages. In R2020 the custom ROS messages could be generated on one computer running MATLAB, shared with another computer running MATLAB, and the shared files would work with the new computer with some simple MATLAB path updates. This was ideal for the purposes of developing and sharing the toolbox. ROS custom messages for MATLAB are generated and stored in the toolbox, the students would simply download the toolbox, run a configuration script which would update their MATLAB path, and they would be setup to start working with the Sphero RVRs from their MATLAB. In R2021, the process to generate MATLAB custom ROS messages was modified so that messages generated on one machine could not be copied and used on a different machine. Each computer would have to use MATLAB to generate their own set of MATLAB custom ROS messages. This is not something that can be done by each student running a simple script as it has the prerequisite that each student download and install three separate software: Python 2.7, CMake, and a C++ compiler. We found this to be an extremely time consuming in-class process as we encountered a multitude of student computer configuration issues. In addition to each operating system (Windows, Mac OS, Linux) having unique instructions to follow, other configuration differences such as age of computer and OS, Firewalls, and student computer user names with symbols or spaces all required special mitigation. All of these issues relating to MATLAB generating files for the custom ROS messages resulted in significant class time being taken from working on the projects and put toward just allowing the students to setup the toolbox.

To address these issues we developed a MATLAB-to-action-server interface node. Instead of custom messages being sent from MATLAB directly to the action server, a standard string message, that is built into MATLAB, is sent to a topic. The string messages contain all of the information needed to construct an action server custom message. The MATLAB-to-action-server interface node subscribes to this topic. Whenever a message is sent from MATLAB to the topic, the interface node receives the message, parses out the relevant pieces of data, constructs an action server custom message, and sends the message to the action server.

The second problem that arose was due to our Wi-Fi not being able to handle the continuous streaming of all the data from over thirty robots simultaneously. Having this multitude of robots streaming such large amounts of data resulted in students' robots not responding to some or all commands, and in some cases, the overload resulted in students' robots becoming disconnected from their MATLAB session entirely.

The clear fix for this was to structure data broadcasting with a request/response instead of a constant stream of data. We considered using a service for this process, but that, again, would result in MATLAB needing to generate files for custom ROS messages. Using an interface with a service would not be ideal as it would result in large amounts of data being transferred through the interface. This extra transfer of data could slow down the student laptop to robot communication. Instead, we implemented a node that emulated a service with a pair of topics that act as a request and response. MATLAB sends a standard, built-in string message to the request topic specifying what data the student wants, the node samples the sensor, and sends the information to MATLAB via the response topic.

Motor Control Development		
Approach	**Problem**	**Solution**
Action server with custom ROS messages	R2021a disallowed pre-generated MATLAB custom ROS messages	Create interface on RPi

Sensor Control Development		
Approach	**Problem**	**Solution**
Sensors live stream on topics	Wi-Fi cannot support data streams	Implement request/response data services
Implement request/response data services	Need custom ROS messages	Emulate service with request and response topics

Table 1: Evolution of implementation challenges and solutions.

4 Discussion

In this new toolbox combining MATLAB with the Sphero API, student teams practice programming fundamentals in a powerful, all-in-one but approachable platform. The RPi control system allows limitless expansion for students to get creative with additional software, sensors, and components. Coupled with MATLAB, students can readily perform speech and command recognition, as well as image processing for object, text, and human recognition. Further, MATLAB's visualization and computational tools allow for practice and implementation of other course learning objectives, including technical communication and numerical analysis. ROS also opens up many creative possibilities since, by design, it accommodates communication between multiple robots per student and multiple students per robot.

It is significant that we discovered the unanticipated hurdles that came from Mathworks substantially changing its ROS capabilities between releases. Successful implementation of custom ROS messages is not trivial when accommodating a wide variety of user machines. By implementing the toolbox in our course with hundreds of students, we had tremendous testing capability for our beta version as we discovered we needed many workarounds for user computers of different OS, manufacturers, partitions, and security settings that were not discovered when first testing the beta version in our initial set of eight different instructor computers. Further, we gained a good sense of the robustness required of the hardware peripherals after numerous, unintentional impact and drop tests.

Further toolbox development is planned for easier implementation of additional I2C sensors, sending data to the LCD screen, obtaining pos-vel-acc data to add further context to numerical methods labs, and utilization of the IR send/detect capabilities for robot-to-robot interaction. The software will also be continually refined to mitigate observed sources of frustration encountered by students implementing the toolbox. We are eager to observe the results of our most recent efforts to mitigate installation anxiety and network traffic. Once all initial toolbox development goals are achieved and the robots are fully implemented in the course, student feedback and assessment data will be collected and analyzed. All code and documentation are freely available to the public via GitHub.

References

[1] R. M. Felder, L. K. Silverman, *et al.*, “Learning and teaching styles in engineering education,” *Engineering Education*, vol. 78, no. 7, pp. 674–681, 1988.

[2] M. Lumsdaine and E. Lumsdaine, “Thinking preferences of engineering students: Implications for curriculum restructuring,” *Journal of Engineering Education (Washington, D.C.)*, vol. 84, no. 2, pp. 193–204, 1995.

[3] A. Behrens, L. Atorf, R. Schwann, B. Neumann, R. Schnitzler, J. Balle, T. Herold, A. Telle, T. G. Noll, K. Hameyer, and T. Aach, “MATLAB meets LEGO mindstorms—a freshman introduction course into practical engineering,” *IEEE Transactions on Education*, vol. 53, no. 2, pp. 306–317, 2010.

[4] J. Summet, D. Kumar, K. O’Hara, D. Walker, L. Ni, D. Blank, and T. Balch, “Personalizing CS1 with robots,” *SIGCSE Bulletin*, vol. 41, no. 1, pp. 433–437, 2009.

[5] T. R. Hamrick and R. A. Hensel, “Putting the fun in programming fundamentals-robots make programs tangible,” in *2013 ASEE Annual Conference & Exposition*, pp. 23–1012, 2013.

[6] R. V. Aroca, F. Y. Watanabe, M. T. De Avila, and A. C. Hernandes, “Mobile robotics integration in introductory undergraduate engineering courses,” in *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, pp. 139–144, IEEE, 2016.

[7] C. Pomalaza-Raez and B. H. Groff, “Retention 101: Where robots go...students follow,” *Journal of Engineering Education (Washington, D.C.)*, vol. 92, no. 1, pp. 85–, 2003.

[8] M. M. McGill, “Learning to program with personal robots: Influences on student motivation,” *ACM Transactions on Computing Education*, vol. 12, no. 1, pp. 1–32, 2012.

[9] J. M. Esposito, O. Barton, and J. Kohler, “Matlab toolbox for the irobot create.” www.usna.edu/Users/weapsys/esposito/_files/roomba.matlab/, 2011.

[10] W. Schleter and A. Biegalski, “Implementing a robotic programming project in a first semester “programming for engineers” course.,” in *Proceeding of the 7th Annual First Year Engineering Experience Conference*, 2015.

[11] Stanford Artificial Intelligence Laboratory et al., “Robotic operating system.” https://www.ros.org, 2021.

[12] E. C. Pearce, “Advanced autonomous kit for sphero rvr assembly guide.” https://learn.sparkfun.com/tutorials/advanced-autonomous-kit-for-sphero-rvr-assembly-guide, 2021.