Teaching Embedded Systems with a Robotics Theme using ROS on the Raspberry Pi

Christopher J. Lowrance and Dominic M. Larkin

***Abstract*—Many courses teaching the principles of embedded systems primarily focus on programming microcontrollers. However, the domain of embedded systems is a broad domain that encompasses other processor types including single-board computers (SBCs) with operating systems. SBCs are often overlooked in the curriculum of embedded systems despite their ability to meet the computational requirements of more sophisticated applications, such as robot controllers and network routers. To address this gap, we restructured our course to include coverage on SBCs while also maintaining some lessons and projects devoted to the register-level programming of microcontrollers. This paper discusses how we integrated SBCs, in the form of the Raspberry Pi 3 (RPI3), into our project-based, undergraduate course. Furthermore, the paper reviews how we introduce the fundamentals of the Robot Operating System (ROS) using a series of projects that are now feasible because of the incorporation of SBCs and a Linux distribution into our curriculum.**

***Index Terms*—embedded systems curriculum, single-board computers, Raspberry Pi, teaching robotics, ROS**

I. Introduction

An embedded system is a special-purpose system in which the computer is encapsulated inside of it and is dedicated to a set of specific functions [1], [2]. The types of processors within an embedded system can vary, and often times, they are either a microcontroller unit (MCU), a field programmable gate array (FPGA), or a microprocessor on a single-board computer (SBC). Furthermore, the number of applications which can be classified as an embedded system is extensive and includes control systems, networking products, robots, commercial appliances, and many others [3]. The diversity of this field of study makes it challenging to cover within a single undergraduate course.

Nevertheless, many universities offer an upper-level undergraduate course titled as *'embedded systems'* or with similar wording, but because of its diverse nature, elements of the subject are often taught in a series of courses [3], [4]. Most of these introductory courses at the undergraduate level seem to focus on the bare-metal programming of microcontrollers without an operating system, while some others elect to include more advanced topics including real-time operating systems (RTOS) and design models for FPGAs and hardware description. However, limiting the course to microcontrollers and FPGAs overlooks a significant portion of embedded systems that use processors running an operating system (OS) and the additional capabilities that this enables.

Today, Linux distributions exist as the OS in many embedded systems located inside of aeronautical equipment, robots, and others [5]. These embedded computers use an operating system to abstract away the lower-level hardware and allow developers to focus on building more sophisticated, intelligent and complex behaviors in systems. The additional resources of an SBC and the OS typically allow more advanced behaviors not feasible with simple microcontrollers. Some applications require more processing power and the concurrent execution of processes that microcontrollers cannot support. Consequently, we felt it was essential to also expose our students to SBCs and a Linux distribution in addition to the classical subject area of microcontroller register-level programming.

The challenge we faced was how to integrate these topics into a semester-long course. Additionally, we considered the need to prepare students for building robotics projects in their capstone design course taken during their senior year. After some careful planning regarding the topics to cover and the amount of time to devote to them, we revised our embedded systems course at the United States Military Academy (USMA) to meet our goals. The contribution of this paper is a detailed discussion on the makeup of our course and how it is structured to cover the programming of MCUs and SBCs with an application emphasis on robotics.

Our intent with this approach is to offer broader exposure to the field so that in the future, our students have the necessary skills for selecting the appropriate processors and are capable of programming them to meet design specifications. There were other motivating factors that stimulated the reorganization of our course. In recent years, we witnessed that several of our senior design projects were starting to incorporate SBCs into their designs either because they demanded more computational power than supportable by most MCUs, or they preferred the layers of abstraction offered by an OS and the rich assortment of high-level application libraries afforded to processors running Linux. For instance, some of these capstone projects included the use of SBCs for autonomous robot navigation planning [6], robot estimation of battery life using machine learning [7], and the detection of security anomalies in the smart grid [8].

A course in embedded systems lends itself to the opportunity for hands-on learning and the presentation of exciting real-world examples. With the projected growth of robotics, arguably robots provide a great example and opportunity to expose the students to a field with plenty of growth and potential. Additionally, in the past, robots have proven to

C. J. Lowrance and D. M. Larkin are with the Department of Electrical Engineering and Computer Science, United States Military Academy, West Point, NY 10996 USA (e-mail: christopher.lowrance@usma.edu and dominic.larkin@usma.edu).

capture the attention of students and motivate them in the classroom [3]. Furthermore, we consistently offer a few senior design projects every year that are related to robotics, and we wanted the course to better prepare the students for working on these types of systems. Hence, we based all of the projects in the latter half of the course on the application of robots in different scenarios. Because SBCs are used in this portion of the course, we go beyond simple microcontroller-based robots and introduce the students to a modern design framework for multiprocessing. The framework we use is known as *Robot Operating System* (ROS). Some of the services provided by ROS include hardware abstraction, message-passing between concurrent processes, package management, and visualization tools [9].

Our embedded systems course is part of a broader sequence of electives that make up the robotics depth option within the electrical engineering curriculum. The course is primarily taken by electrical engineers (EE) in their junior year, but may also be selected as an elective by computer science (CS) majors. The only prerequisite is computer organization, which is taken by both EE and CS majors and covers assembly and C programming. The computer organization course also exposes the students to the Linux command line and file system, primarily as a result of compiling their C programs using GCC. Because embedded systems is an elective for EEs, our computer organization class also covers a short embedded systems block where they use the Arduino microcontroller. Hence, the EE students enter the embedded systems course already having exposure to the Arduino. Another course that is part of the core curriculum at USMA is titled "Introduction to Computing and Information Technology", and it covers the basics of programming in Python.

II. Related Work

Embedded system courses generally center on a series of hands-on exercises and projects. However, when comparing curriculums at different universities, the specific topics emphasized can vary because of the wide range of applications and hardware used in embedded systems. Ibrahim et al. surveyed different teaching approaches and found them to be diverse, but concluded that programming a microcontroller is an essential topic in an embedded systems course [2]. Jamieson concluded that the open-source Arduino microcontroller, which abstracts register-level programming through its high-level integrated development environment (IDE), is an acceptable processor for teaching embedded systems because of the other complexities and challenges that students encounter during projects [10]. In another course, Koopman et al. discovered that enforcing students to follow a design methodology increased the chances that their project worked before deadlines [3]. In a class by Lee et al., three different methods of design are emphasized including programming microcontrollers at the bare-metal (i.e., no OS), RTOS, and model-based (e.g., LabView) levels [11]. In a Massive Open Online Class (MOOC), Valvano et al. emphasized microcontroller programming and used a bottom-up approach to build competencies for later opened-ended projects [1]. In the second of a series of microcontroller courses, Beavis et al. focus on teaching the advanced principles of RTOS by having student design a robot that must simultaneously navigate through a maze, detect artificial landmines and communicate with other robots [4]. Danowitz et al. use the Raspberry Pi 3 (RPI3) and an autonomous robotics project to introduce the principles of Linux and embedded programming to new engineering students in a short summer course [12]. With the growth of smartphones and single-board computers, Linux is becoming the operating system of choice for many embedded products, and as a result, some professors, such as Wang et al. and McLoughlin and Aenderoomer, have adapted their courses to cover concepts related to embedded Linux [5], [13].

III. Course Structure: Teaching Microcontrollers, SBCs, Linux, and ROS

A. Structural Overview of the Course

Project-based learning (PBL) is an active form of learning that is known to promote better retention and higher-order thinking in students than traditional lecture-style teaching [14]. Thus, our course takes a PBL approach and has the students progress through a series of mini-projects. Because the projects are the central focus, they account for 80% of the students' grades, while the other 20% is based on two written exams that are strategically delivered at transition points in the course. Instead of assigned homework problems, students are expected to work on their projects outside of the time allotted in class. Each project concludes with a product demonstration and a written report.

The course consists of three primary phases and a total of seven projects as illustrated in Fig. 1. The first phase focuses on programming MCUs at the register-level and exploring the features that distinguish them from ordinary processors. Afterward, in the second phase, the course introduces SBCs and uses them to familiarize the students with Linux and reacquaint them with the Python programming language. Finally, the course concludes with projects that utilize ROS to expose the students to the publisher and subscriber paradigm and an open source toolset for building more complex robotic designs. By exposing the students to multiple hardware devices, our intent is for the course to offer more breadth into the subject so that the students are better prepared to select the best processor for their applications.

B. Phase I: Programming Microcontrollers at the Register-Level

The first 15 of the 40 lessons in the course are devoted to the study of MCUs. The focus during this period is on the bare-metal programming of MCUs using the C programming language. Instead of relying on the layers of abstraction from an operating system, students learn to program at the register-level. The primary focus is on configuring the MCU to use its input/output peripheral devices (e.g., digital pins, analog-to-digital converter, serial communication ports, timers, and others) in the context of an application coded as a state machine that is event-driven. On the other hand, the topic of real-time operating systems (RTOS) is omitted because it is

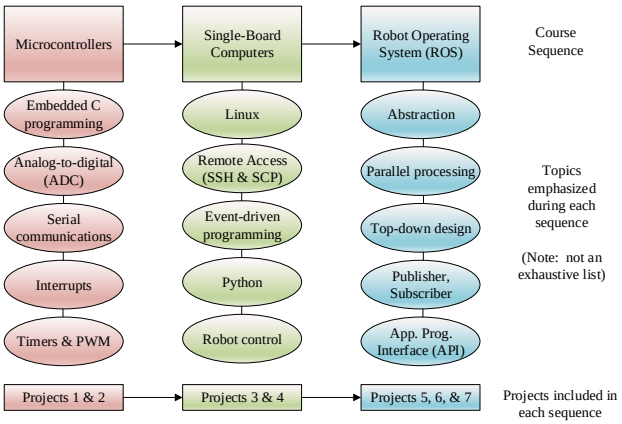

Fig. 1. Illustration of course sequence and some of the topics emphasized in each phase.

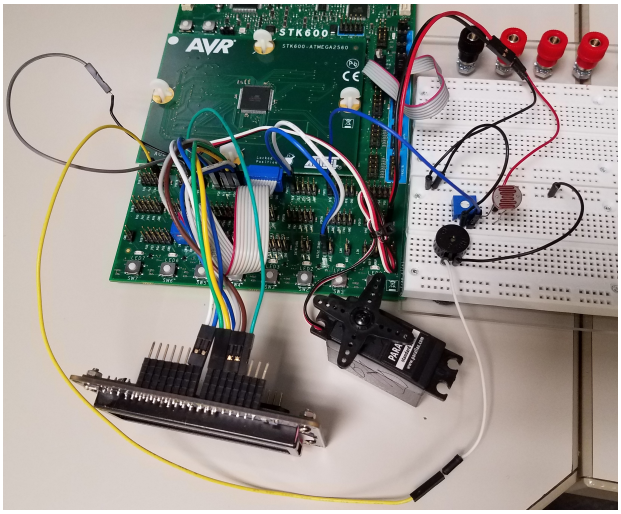

Fig. 2. Picture showing the hardware used in the MCU project focused on the prototyping of a digital safe. The pushbuttons on the STK600 are used for entering the combination, the servo simulates a locking mechanism, and the LCD and piezoelectric speaker provide feedback to the user.

considered a more advanced issue [4]. Instead, time in the lesson schedule is devoted to other topics as discussed in the subsequent sections.

Classroom lessons typically start with an overview of a particular microcontroller feature and then transition to programming that feature using the datasheet. Afterward, the students use the remaining time allotted for in-class exercises utilizing a program example provided to the students. The code in these examples is either entirely or mostly functional so that the students can complete the task in the time allotted. The intent is to familiarize them with using the microcontroller feature and building their confidence in programming the device using the software environment.

The course uses the STK600 development board along with the Atmel ATmega 2560. Microcontrollers from the AVR family, including the ATmega 2560, are used in the popular open source Arduino microcontroller. Due to the students being exposed to the Arduino in an earlier class, it makes sense that we continue to explore the microcontroller at a greater depth, as well as show them an alternative way for programming the device outside of the Arduino IDE. This course uses the Atmel Studio 7 as the IDE.

We assign two projects during this MCU block. The first project serves as an orientation to low-level microcontroller programming and Atmel Studio. It consists of a series of introductory exercises. The project starts by relating the high-level Arduino IDE to Atmel Studio by investigating the potential drawbacks and limitations of abstraction, especially regarding precision and customization. Afterward, the students learn to program a different Atmel microcontroller outside of the STK600 and on a breadboard using an AVR programmer, not the STK600 one. The exercise culminates with the students developing a state machine program that changes the blinking pattern of the LEDs on the STK600 based on a push button interrupt.

The second project is more complicated and requires the students to integrate previously studied concepts. In this project, students design and build a prototype for a digital safe using the STK600 along with the additional hardware shown in Fig. 2. In order to build the prototype, the students must serially interface the MCU with a liquid crystal display (LCD), use Pulse Width Modulation (PWM) to control a servo motor, use an analog-to-digital converter to control the backlight of the LCD, use interrupts to detect button presses, and modulate a digital pin to excite a piezoelectric speaker. Given its numerous components, we emphasize incremental building and code encapsulation (via header files) during this project.

C. Phase II: Transitioning to Linux, Single-Board Computers, and Robotics

In the second phase of the course, we introduce single-board computers (SBCs) and use them in the remaining projects. By switching over to an embedded processor that supports a Linux distribution, we can abstract some of the low-level details studied in the first part of the course, and instead, focus on leveraging the more powerful processor in applications that require more sophistication than microcontrollers can support. But before working on complex applications, the projects in this phase of the course are more straightforward and focus on getting the students comfortable with the hardware, the Linux environment, and the Python programming language. Concerning Linux familiarity, the emphasis is placed on navigating the operating system command line, remoting into the SBC from a workstation via secure shell (SSH), copying files remotely via secure copy (SCP), using command line text editors, and other common Linux features.

The Raspberry Pi 3 (RPI3) was chosen as the SBC to use in the course for a couple of reasons. First, it is an inexpensive computer that cost under $40. Secondly, the device has attracted much attention in recent years, and there are several forums available online to help students get started with the platform and assist with troubleshooting when necessary. In terms of programming languages, we chose Python for the remainder of the course because of its usability, rich assortment of libraries, and the fact that the students learned to program with it in a previous class. Python has also proven

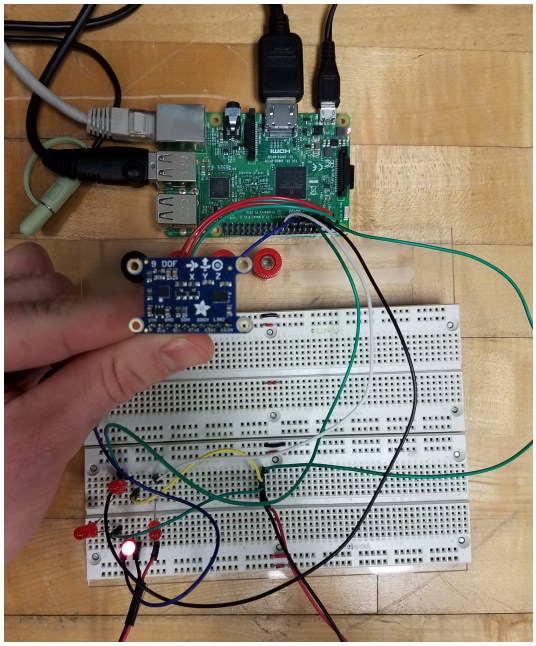

Fig. 3. Picture showing the hardware used in the initial SBC project that develops a driver for an IMU sensor and uses LEDs to indicate its orientation.

Fig. 4. Image showing the robot path-following project.

to be a popular language for rapid prototyping which is appropriate for our projects. At this time, we typically highlight the advantages and disadvantages of compiled languages, such as C, and interpreted languages like Python.

There are two projects associated with this transitional and preparatory phase of the course. The first project serves as an introduction to the RPI3 and the concepts discussed above. It consists of a series of familiarization and competency building exercises that prepare the students for the final element of the introductory project, which consists of creating a driver for an inertial measurement unit (IMU) sensor. The projects start with some orientation exercises that involve building a couple of "hello world" type circuits using LEDs and push buttons that interface with the RPI3's general purpose input/output (GPIO) pins. Afterward, there is an exercise that investigates the RPI3's ability to generate precise PWM signaling in software, and as the students discover, SBCs do not perform this function well because of the lack of PWM hardware, which results in persistent signal jitter. Finally, the students conclude the project by writing their driver for an IMU. By doing so, the students leverage concepts studied in the first part of the course, including using a datasheet to lookup register settings, performing bitwise operations, and configuring serial communications. In the process, the students gain an appreciation for abstraction and the ability to import a library that helps set up and negotiate the I^2C protocol which is required to interface with the sensor. As a product demonstration, the students convert the magnetometer data into a heading that is then used to illuminate one of four LEDs indicating north, east, south, and west directions. Fig. 3 shows an example setup of this project.

The second project in this phase marks the beginning of the robotics emphasis in the course. In this project, the students must program the robot shown in Fig. 4 to stay within the boundaries of a curved path outlined in colored tape. A camera known as Pixy [15] is mounted toward the front of the robot, and the students use a Python application programming interface (API) to detect the colored tape. A motor controller peripheral to the RPI3 is connected to the GPIO pins to generate clean PWM signaling that is free from jitter. PWM is used to control the robot's speed and steering, and the students develop a control algorithm that has the robot autonomously navigate within the lined path based on the detected positions of the colored tape.

The project serves multiple purposes and provides the students with the following: familiarization with the robotic platform used in the remainder of the course, an example of PWM motor control, practice using an API, and reinforcement of state machine programming. Videos showing the students' work on this project, as well as the other robot-based projects discussed in this paper, are available on the West Point Robotics YouTube channel (see the "EE487 Embedded Systems" playlist) [16].

D. Phase III: Introducing ROS for going beyond basic MCU-based robots

In the final phase of the course, we introduce the fundamentals of ROS. ROS is an open-source, meta-operating system that can be installed with Linux and on the RPI3. The primary emphasis during this portion of the course is the publisher-subscriber framework of ROS. This framework facilitates the decomposition of complex designs into an interconnection of simpler nodes (or concurrent processes) that communicate via message passing.

A total of three robotic projects are completed using ROS. The first project serves as an introduction and involves controlling the robot shown in Fig. 4 using an Xbox joystick

that is plugged into a laptop. The ROS framework enables the wireless message-passing between a computer and RPI3 on the robot. By completing the project, the students gain an appreciation of the abstraction and message-passing framework provided by ROS. For instance, students leverage the ROS repository to download and install a device driver for the joystick without needing to write any code. Furthermore, with the modification and addition of a few lines of code to a ROS template script, they were able to configure a subscriber node to execute on the RPI3 that would listen to messages being published by the joystick over the wireless network. The code written by the students in this subscriber node interprets the joystick commands and maps them to meaningful PWM signals to drive the robot's motors.

The second ROS-based project involves writing a state machine program for the same robot to autonomously navigate down the hallway of our building using a Hokuyo URG-04LX Scanning Laser Rangefinder. Again, the students use the ROS repository for downloading and installing a device driver for the LIDAR. For the state machine navigation code, most students divide the laser returns into sectors and then calculate a single average distance for objects in each sector. By reducing the size of the laser scan to only a few sectors of concern, most students successfully construct a state machine to control the steering of the robot based on which sectors detect an object near the robot.

The final ROS project has the students develop an algorithm for the robot to navigate between outdoor GPS waypoints autonomously. A driver for the GPS sensor is installed from the ROS repository. However, there is no driver in the repository for the IMU used in the course. Intentionally, the same IMU that the students created a driver for in the previous block is repurposed in this project. Therefore, at this point, the students do not need to write new code to interface with the IMU and only have to convert their previous script into a ROS publisher. By publishing the sensor data in ROS, it is made available for other nodes to subscribe to, such as the node responsible for controlling the robot towards a waypoint. Once the students successfully get their robot navigating between waypoints, they begin to explore the topic of odometery. Specifically, they use the onboard wheel encoders of the robot to detect displacement, and when combined with the IMU data, they can begin to localize the robot in an odometery frame which is also compared to the robot's perspective based on GPS.

IV. Student Feedback on Course Organization

A survey was conducted to determine students' perceptions about the structure of the course. A total of 17 students from the last two iterations of the newly structured course responded to the survey. There were ten statements in the survey. All survey statements used a Likert scale for agreement with five equally-spaced response options from strongly disagree to strongly agree. Fig. 5 is a bar chart summary of the results. The response options in the figure are given numerical weights for the purpose of plotting: Strongly Disagree = 1, Disagree = 2, Neutral = 3, Agree = 4, and Strongly Agree = 5.

In general, we can conclude from the survey that most students liked the project-based model and enjoyed working on

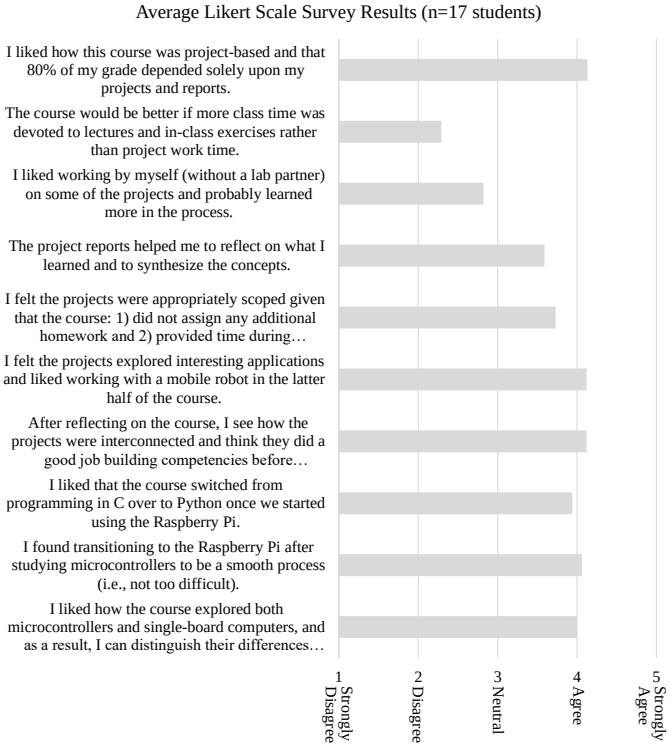

Fig. 5. Survey Results.

applications, including those involving robots. In our course, we elected to make some of the projects individual assignments because we had sufficient equipment, and we wanted to maximize the learning experience for each. On the other hand, for more complicated projects, we allowed groups of two or three students to work together. However, as the third survey response shows, it appears that students feel they learn better in a group setting. In the fourth statement, we intended to investigate the value of having students submit written reports at the conclusion of every project. From their responses, the reports are beneficial because they have them reflect on the project and synthesize the concepts. However, during the next iteration of the course, we are considering replacing some of these written reports with oral presentations.

One of our primary concerns that we wanted to sense in the survey was whether the students felt overwhelmed with the complexity of the projects, the amount of work involved, and the nature of the transition in hardware and software. Overall, we discovered that most students felt that it was manageable and that the hardware and software transition from MCUs to SBCs was not too challenging. But in terms of these questions, the weakest agreement was concerning the scoping and complexity of the projects and whether sufficient time was provided to the students. However, to make the applications meaningful and somewhat challenging, we feel the level of complexity and the time provided is roughly sufficient.

V. Conclusions

This paper explains how the structure of our embedded systems course bridged the gap between register-level program-

ming of MCUs and using higher-level programming tools on SBCs with an OS. The early use of the STK600 development board provides the students the chance to learn register-level programming and the intricacies of microcontroller hardware. Afterward, they are exposed to more abstraction with an SBC, an operating system, and Python programming language. Finally, they learn about ROS and apply the framework in a series of increasingly complex robot projects.

We believe that incrementally adding the layers of abstraction in the projects increased understanding of the course material and prepared them to make design decisions. Students seemed to achieve a good understanding of the capabilities and limitations of the various embedded systems in which the class uses. The survey results indicated that the transitions between the types of hardware and software, as well as the complexity of the projects, were manageable. We attribute this success to the strategic organization of the course, including its projects that gradually introduce the students to the platforms and concepts before advancing in complexity.

VI. Acknowledgments

This work was prepared by an officer or employee of the United States Government as part of that person's official duties and, therefore, is a work of the United States Government. In accordance with 17 U.S.C. § 105, no copyright protection is available for this work in the United States.

References

[1] J. W. Valvano, R. Yerraballi, and C. Fulton, "Teaching Embedded Systems in a MOOC Format", *ASEE Annu. Conf. Expo.*, Jun. 2016.

[2] I. Ibrahim, R. Ali, M. Z. Adam, and N. Elfidel, "Embedded systems teaching approaches & challenges", in *IEEE 6th Conference on Engineering Education*, 2014, pp. 3439.

[3] P. Koopman et al., "Undergraduate Embedded System Education at Carnegie Mellon", *ACM Trans Embed Comput Syst*, vol. 4, no. 3, pp. 500528, Aug. 2005.

[4] P. Beavis et al., "Using Robots To Teach Complex Real Time Embedded System Concepts", *ASEE Annu. Conf. Expo.*, pp. 10.1425.1-10.1425.12, Jun. 2005.

[5] I. McLoughlin and A. Aendenroomer, "Linux as a teaching aid for embedded systems", in *International Conference on Parallel and Distributed Systems*, 2007, vol. 2, pp. 18.

[6] P. Maxwell, D. Larkin, and C. Lowrance, "Turning Remote-Controlled Military Systems into Autonomous Force Multipliers", *IEEE Potentials*, vol. 32, no. 6, pp. 3943, Dec. 2013.

[7] S. J. Miller et al., "A dynamic ensemble for estimating state-of-charge of interchangeable robot batteries", in *IEEE MIT Undergraduate Research Technology Conference*, 2017, pp. 15.

[8] K. Candelario, C. Booth, A. S. Leger, and S. J. Matthews, "Investigating a Raspberry Pi cluster for detecting anomalies in the smart grid", in *IEEE MIT Undergraduate Research Technology Conference*, 2017, pp. 14.

[9] *ROS/Introduction - ROS Wiki.*, [Online]. Available: http://wiki.ros.org/ROS/Introduction. [Accessed: 28-Mar-2018].

[10] P. Jamieson, "Arduino for Teaching Embedded Systems. Are Computer Scientists and Engineering Educators Missing the Boat?", *Proc FECS*, vol. 2010, pp. 289294, 2010.

[11] E. A. Lee, S. A. Seshia, and J. C. Jensen, "Teaching Embedded Systems the Berkeley Way", in *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*, New York, NY, USA, 2013, pp. 1:11:8.

[12] A. Danowitz, B. Benson, and J. Edmonds, "Teaching Systems and Robotics in a Four-Week Summer Short Course", *ASEE Annu. Conf. Expo.*, Jun. 2017.

[13] K. L. Wang, C. S. Cole, T. Wang, and J. Harris, "An Effective Project-Based Embedded System Design Teaching Method", *ASEE Annu. Conf. Expo.*, pp. 23.160.1-23.160.9, Jun. 2013.

[14] J. Savery, "Overview of Problem-based Learning: Denitions and Distinctions", *Interdiscip. J. Probl.-Based Learn.*, vol. 1, no. 1, May 2006.

[15] *Pixy (CMUcam5) Charmed Labs.*, [Online]. Available: http://charmedlabs.com/default/pixy-cmucam5/. [Accessed: 28-Mar-2018].

[16] *West Point Robotics*, YouTube. [Online]. Available: https://www.youtube.com/channel/UC5-d060L_I9vaGRr9NV0npg. [Accessed: 13-Mar-2018].