The Bumblebee: A Robot Controller Board for STEM Education

Lana Hodzic, Kevin Ly, Aaron W. Keen, John S. Seng
Computer Science Department
Cal Poly State University, San Luis Obispo

Abstract

This paper provides a case study description of the Bumblebee robot controller. We designed the Bumblebee as a controller board that provides hardware and software support for elementary aged students interested in building robots and learning about electronics. When teaching elementary aged students, one is often limited to a selection between Arduino type hardware and LEGO Mindstorms kits. Arduino hardware can be limiting for a new student because the boards are not tolerant of wiring errors. LEGO Mindstorms kits can be prohibitively expensive and can be limiting in the number of sensors that can be attached.

The hardware for the Bumblebee is designed to allow interfacing with readily available Arduino shields while providing more usability features such as an LCD screen. One of the important aspects of the Bumblebee controller is fault tolerance. Students can make several different types of connection errors and the board will handle these faults without damage.

Buzz is a companion programming language that is designed for elementary age students to easily program the board. We developed a front-end preprocessor that takes in Buzz code and converts the source into C which can be compiled with available compilers.

We find that the Bumblebee is suited for university level instruction as well. The fault tolerant aspects are extremely practical and we allow students to program the boards in C. This board was successfully used in a university-level robotics class this Fall 2015 and we outline the experience.

We have tested the Bumblebee with elementary-aged students in workshop type environments and find that the opportunity to use a breadboard and build electrical circuits keeps the students highly engaged and excited to learn.

Introduction

Robotics and electronics are excellent motivators for teaching engineering principles to students. The excitement of seeing something move or seeing a circuit function encourages students to learn more about these fields.

Elementary school children in particular have interests in these areas, but some readily available options may be technically challenging for their age group or may be cost prohibitive. In this work, we describe the development of the Bumblebee controller board. The Bumblebee board is designed for elementary-aged students to build robotics and electronics projects. This age group requires a board that is easy to program, has simple connectors, is tolerant of wiring faults, and can easily run on batteries.

This paper is organized as follows. We first describe work that is related to the Bumblebee board in both the hardware design and in software capabilities. Next, the hardware details of the board are covered as well as some of the differences between the readily available Arduino Uno platform. The software overview section describes the Buzz programming language that has been developed for the Bumblebee. Finally, we describe the testing experiences we have in using the Bumblebee with elementary school students and college students as well.

Related Work

The Bumblebee board falls into the class of 8-bit microcontroller boards that has been greatly popularized by the Arduino Uno and derivatives. These boards are commonly based on the Atmel AVR architecture. Some boards that have been introduced over the past few years which are intended for robot designs include the Digilent chipKIT Mx3 (Digilent, 2016) and the Sparkfun RedBoard (Sparkfun, 2016). The Bumblebee is most similar to the MIT HandyBoard (Martin, 2001) which featured an on-board LCD display, on-board motor drivers, and a pre-built software library.

The fault tolerant design of the Bumblebee is similar in spirit with the commercial Ruggeduino (Rugged Circuits, 2016). The Ruggeduino is an Arduino Uno compatible design which builds in many fault tolerant aspects to prevent damage due to user error.

The Buzz programming language and the Lego Mindstorms kit (Lego, 2016; Hixon, 2007) fulfill the same niche of robotics programming directed at a younger audience. The Lego Mindstorms kit uses graphical blocks in a drag-and-drop fashion to allow students to build their program. Unlike the Buzz language however, the Mindstorms kit is not based on event-based programming. There does not exist an explicit loop that runs forever or triggers that activate when programmed conditions are met. The Mindstorms kit can also be prohibitively expensive for schools operating on a low budget with large classes.

Other efforts to encourage younger students to program include the free programming language Scratch (Maloney et al., 2010). Scratch also uses color coded blocks that students drag-and-drop to create their programs. To provide feedback to the users, instead of controlling a robot, the student manipulates sprites in a window and watches them move. Buzz on the other hand emphasizes textual code programming instead of drag-and-drop style programming. In

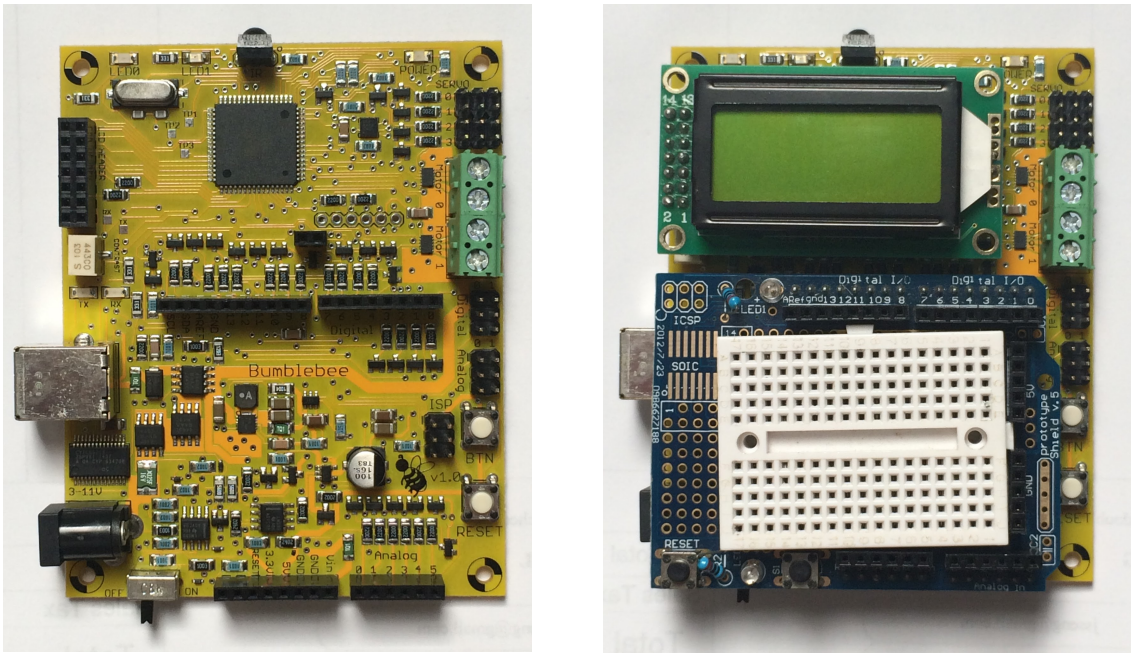

Figure 1: Bare Bumblebee board (left), Bumblebee with LCD and breadboard shield (right)

contrast to a computer graphic, students will be able to control the Bumblebee board coupled with circuitry or robot hardware, allowing a tangible experience of the actual consequences of their code.

Hardware Overview

The hardware design philosophy of the Bumblebee was to create an Arduino shield-compatible board that could easily integrate with existing robot hardware. The target audience was elementary school students, so the hardware needed to be tolerant of wiring faults, operate at a low voltage, and require minimal configuration. In this section of the paper, we list the hardware features and implementation details.

Hardware Features

The hardware functionality of the Bumblebee is listed below:

- Arduino Uno shield compatible
- Removable 2 line by 8 character backlit LCD
- 2 DC motor ports (1.8A maximum current each)
- 4 RC servo ports
- 3-axis accelerometer

- IR receiver for decoding remote control signals
- 16MHz Atmel ATmega645A (64KB flash, 4KB RAM)
- 2 software-controlled LEDs
- 1 user pushbutton
- DC power jack (3V to 11V)
- On/off switch

Advantages over Arduino

The standard Arduino Uno (Arduino, 2013), though low cost, has a number of disadvantages when used to teach basic electronics, embedded systems, or robotics (Jamieson, 2010).

Firstly, an Arduino requires an input voltage of at least 7V. This voltage requirement arises from the use of a regulator chip which requires input of at least 2V more than the regulated output of 5V. Because of this requirement, there are many classroom assignments which specify that a separate battery, most often a 9V, is required to power the Arduino while another battery is used to power other devices, such as 4xAA batteries for motors or servos. This 2 battery requirement can be complicated for students to understand and furthermore complicates the overall wiring. The Bumblebee has a voltage input range which extends down to 3V. This allows the board and motors to be powered by 2xAA batteries if necessary while still providing 5V to the other parts of the board.

Secondly, the Arduino maps the first pair of primary digital pins, pins 0 and 1, to the wires that are used in the firmware uploading process. This default mapping often confuses students because the Arduino will often fail during upload operations if there are any sensors or devices connected to digital pins 0 and 1. Students have to temporarily disconnect devices, perform the upload, and then reconnect the devices to test. This process can be cumbersome when uploading firmware to the Arduino. Because the Bumblebee uses a dedicated set of pins for the upload communication, there is no contention issue over those pins.

Thirdly, a major advantage of the Bumblebee over the Arduino is fault tolerance. As part of the learning process, students may incorrectly wire a circuit which can easily damage an I/O pin on an Arduino. Wiring faults such as shorting power to ground or shorting digital I/O pins to ground are common. The Bumblebee is designed to tolerate incorrect wiring of the I/O pins and this allows instructors and students to focus on learning about circuits without worrying about damaging the boards.

Firmware

The Bumblebee uses a modified version of the Optiboot (Optiboot, 2016) bootloader. This is an open source bootloader and is an alternative to the original Arduino bootloader. This bootloader

is Arduino compatible and is flashed onto the primary microcontroller before the board can be used. Modifications to the stock Optiboot bootloader include a built-in testing mode which the user can run regardless of the current state of the user-flashed program. This testing mode allows testing of the inputs, motors, servos, and battery voltage. Testing mode is enabled by holding down the user button while pressing and releasing the reset button.

Hardware Prototyping

The initial funding for the development of the Bumblebee was supported by an internal Cal Poly grant. This grant was used to build a few initial prototypes and eventually the funds were used to do a bulk build of 100 boards.

Software Overview

Programs for Arduino compatible boards, such as Bumblebee, are typically written in the C programming language. With elementary school students as the target audience, we explicitly expect that this audience will not have prior programming experience. The low-level nature of the C programming language coupled with the conceptual overhead of learning both the hardware and software aspects of working with Bumblebee prompted the exploration of alternatives. The outcome of this exploration is the design and implementation of an alternate development language, named Buzz, to support the introduction of robotics programming to elementary school students.

Buzz Programming Language

Buzz is a programming language geared towards elementary school students and is designed for programming in an event-based model. Buzz features a more natural English-based syntax with less syntactic complexity than C. The event-based model allows students to write code segments to react to external stimuli (button presses, updated sensor data, clock ticks, etc.); this style of asynchronous programming allows the student to focus on which actions to take in response to events without concern for the overhead of polling for changes, checking against previous values (for updates), and triggering code.

Buzz programs are textual. The language design intentionally eschewed a drag-and-drop graphical model, like that used in the LEGO Mindstorms kit, to reinforce the link between instructions (even at the high-level of Buzz) in the program and operations by the robot. Instead, the syntax of Buzz mimics that of Python to reduce syntactic overhead and uses English-language like keywords to help children leverage their familiarity with a natural language in understanding the programming language. This decision was made to help abstract out C

```
define thermometer = analogPinIn[0]
define leftServo = servo[0]
define rightServo = servo[1]

when start {
  display "Hello! Machine is starting up"
}

repeat {
  display "Thermometer reading is: " + get thermometer
  set leftServo 15
  set rightServo 75

  display "Temperature in Celsius is: " + ((get thermometer - 32) * 5 / 9)
}
```

Figure 2: Sample Buzz program

concepts that would be confusing and difficult for students to learn when first trying to grasp programming.

The Bumblebee supports, in its default configuration, the following devices: RC servo motors, DC motors, analog sensors, digital sensors, and an on-board button. Some of these devices are write-only (e.g., servo motors and DC motors) and some are read-only (e.g., the analog sensors and the button). For each device, an appropriate set or get keyword is automatically provided by Buzz to support (e.g., setting a voltage or reading an input value).

Figure 2 illustrates the layout of a basic event-driven Buzz program. This program begins by giving names to different devices and continues with the two most basic event handlers. The first, '`when start`', executes once when the program begins while the second, '`repeat`', executes repeatedly for the duration of the program. Within these handlers are examples of simple statements to display text to the screen, to set the servos, and to get thermometer readings.

Figure 3 presents a longer program with examples of more complicated event handling. This example illustrates the use of variables, the definition and invocation of functions, and different forms of event handlers including conditional checks and a general 'changes' check.

```
define tL = motor[0]
define tR = motor[1]

define thermometer = analogPinIn[0]

define gX = gyroscope[X]
define gY = gyroscope[Y]
define gZ = gyroscope[Z]

variable storedTemp = 0

func turnRight() {
  set tL 60
  set tR 30
  return 0
}

func halt() {
  set tL 0
  set tR 0
  return 0
}

func go() {
  set tL 70
  set tR 70
  return 0
}

when get thermometer > 15 {
  halt()
}

when get gX > 100 and get gY > 100 {
  display "Both X and Y > 100"
}

when thermometer changes {
   if get thermometer < storedTemp {
      turnRight()
   }
   storedTemp = get thermometer
}

when get button equals 1 {
  stop
}

go()

repeat {
}
```

Figure 3: Longer Buzz program example. Execution starts from the top.

Development Environment

To aid the novice programmer, the jEdit (jEdit, 2016) programming text editor has been extended with support for the Buzz programming language. jEdit is written in Java, is open source, and contains a plugin architecture which allows different enhancement modules to be written for the editor.

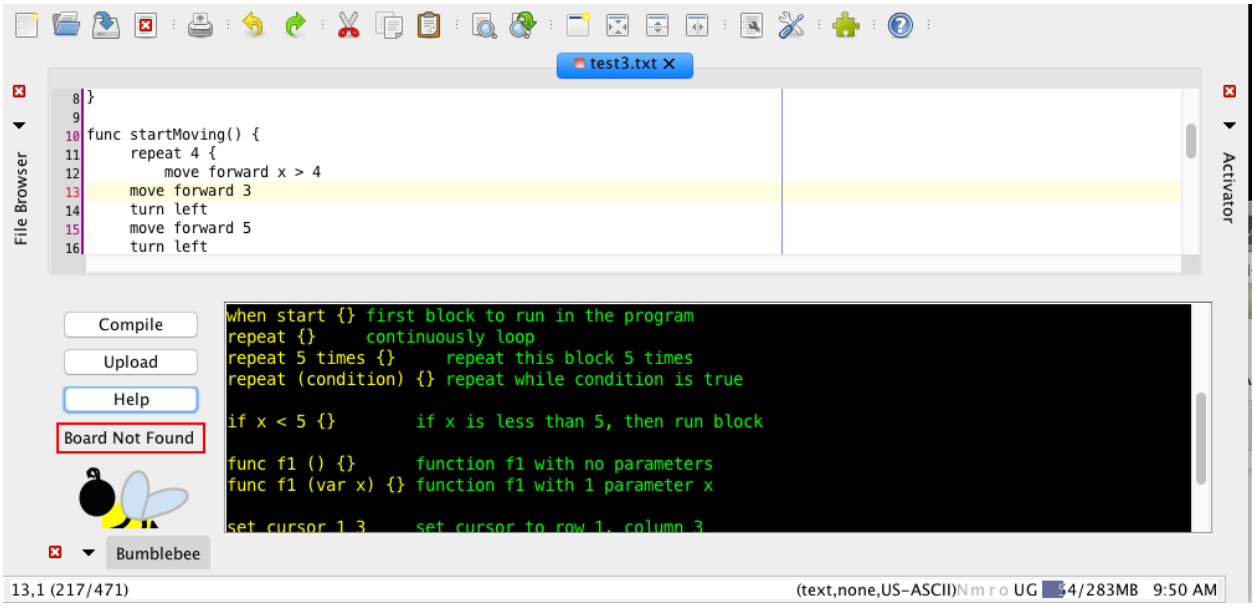

Figure 4: Example jEdit screen with Buzz plugin

The Buzz plugin that we added has features that allow younger students to immediately see what is wrong in their code or their setup of the system. This plugin is shown in Figure 4. There is a "Board Not Found" field with a red border when a board is not properly attached to the computer. The text of this field changes to "Board Found" and the border changes to green when the board is properly attached. For those new to Buzz, there is a *Help* button that will display example code in the output window; this example demonstrates the language syntax and shows how to set up some of the available devices.

The *Compile* button will run the Buzz compiler on the current program. While compiling, if there are errors, the corresponding line number and error explanation are displayed. If there are no problems with the code, then an explicit "No errors" message is displayed (to avoid the "did it do anything" question common with "succeed silently" compilers). This compilation phase also automatically invokes the C compiler to generate an executable that can then be uploaded to the Bumblebee board, thus hiding this detail from the user. The *Upload* button will flash the executable to the Bumblebee board.

Transpiling to C

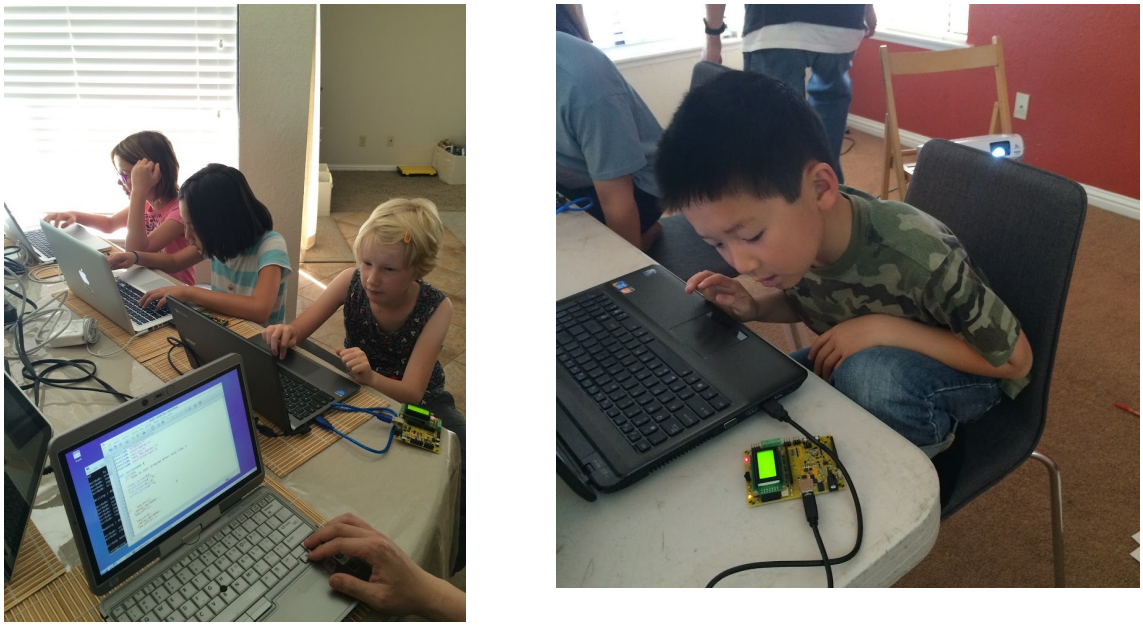

Figure 5: Elementary school students working with the Bumblebee

The Buzz programming language is not, of course, natively supported by the Bumblebee board. The Buzz compiler is actually a Java transpiler that translates the given Buzz program into an equivalent C program leveraging support from a firmware library used to provide a convenient interface with the various devices attached to the Bumblebee board. This translation approach simplifies the development and maintenance of the Buzz compiler and allows Buzz programs to be linked against C libraries during the second (C) compilation step.

Error Checking

Beginning programmers often make syntax mistakes and thus, must contend with compiler errors. Even with the simplified syntax of Buzz, students will undoubtedly make typing mistakes or syntax errors. Though the GCC compiler front-end performs a thorough sweep of the code to ensure syntactic correctness, the error messages produced are not suitable for an elementary school student. For example, in C, if a variable declaration does not end with a semicolon, GCC will output a complex error message.

The Buzz compiler reports more direct error messages about what is missing or incorrect with the syntax. An example of such is: `Identifier name of mach already taken [Line 3]`. Though, at present, one shortcoming of the Buzz compiler is that it will only report at most one error, unlike the gcc compiler which can recover from a syntax error and continue reporting errors in other areas of code. Though this limitation might be considered unacceptable for experienced programmers, in the context of an elementary school activity the one-error-at-a-time model is less intimidating and avoids spurious cascading error messages.

Classroom Experience

The Bumblebee was first tested with elementary school-aged children in the Summer of 2015. This initial deployment of the board was through small summer workshops which consisted of groups of 4-6 students. Each three-hour session consisted of a student working on their own with a laptop and a Bumblebee board plugged into the USB port. In these sessions, we used the board only and did not use a robot base. The session involved a mix of programming in Buzz and C.

The students were first taught how to use the LCD. The LCD consists of 2 lines with 8 maximum characters on each line. Students were instructed on how to move the cursor on the screen, clear the screen, and how to print out messages. Students were also introduced to the concept of time delays in a program. Additionally, a breadboard shield was plugged into the Bumblebee allowing the students to plug wires directly to the board. This allowed the students to wire LEDs directly to individual digital I/O pins. Once the LEDs were connected through the breadboard, programs were written to flash the LEDs at various rates and patterns.

In Fall 2015, the Bumblebee board was used as the primary controller for a college-level introduction to robotics course. The board was coupled with an Ardbot II (Budget Robotics, 2016) robot which is a small robot platform that does not come with a controller. In this course, the students were expected to program in C using a pre-built Bumblebee C library. Students connected individual sensors to the board using either the analog or digital inputs. The robots were motorized using the servos outputs on the board. Assignments in the course involved activities such as line following, implementing a basic neural network similar to Imberman (2003), and Monte Carlo Localization (Dellaert, 1999). The class consisted of students using 18 Bumblebee boards and there were no board failures due to wiring or electrical faults.

Future Work

Future work for the Bumblebee board consists of software work since the hardware design is functional. In terms of IDE development, the jEdit plugin is functional, but the installation procedure requires installing several dependencies and the overall process could be streamlined and automated. In addition, the jEdit plugin can be improved upon to provide even friendlier compiler errors, warnings, and locations of each.

Although the Bumblebee pinout is designed to be Arduino compatible, it is compatible in terms of physical and electrical specifications, but not at the microcontroller software level. The Arduino Uno uses a different microcontroller with a different pinout from the microcontroller on the Bumblebee. That means that in order to function with particular existing Arduino shields,

currently software libraries need to be modified. This future work would be necessary on a per shield basis.

Conclusion

The Bumblebee is an embedded controller board that is designed to teach programming and engineering concepts to elementary school students. We have found the board to be an effective and inexpensive teaching tool for younger students and for university students as well.

The board hardware is designed with features that make it easier to use in the classroom environment. The Bumblebee circuitry is tolerant of wiring faults and is designed to operate from a low voltage battery. A built-in port to connect to commodity LCD displays makes debugging easier and provides visual feedback to students.

In addition to C, the board can be programmed in the Buzz programming language. This programming language supports a simplified syntax and an event-based execution model. Together with the Buzz programming language and the Bumblebee hardware, this platform provides a versatile environment for students to learn about electronics and robotics.

Bibliography

1. S. Imberman. (2003). Teaching Neural Networks Using Lego Handy Board Robots in an Artificial Intelligence Course. *In Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, May 2003
2. Rugged Circuits (2016). *Ruggeduino-SE "Special Edition"*. Retrieved from the Rugged Circuits, Inc. website http://www.rugged-circuits.com/microcontroller-boards/ruggeduino-se-special-edition
3. Digilent (2016). *chipKIT MX3: Microcontroller Board with Pmod Headers.* Retrieved from the Digilent, Inc. website http://store.digilentinc.com/chipkit-mx3-microcontroller-board-with-pmod-headers
4. Sparkfun (2016). *SparkFun RedBoard - Programmed with Arduino.* Retrieved from the Sparkfun, Inc. website https://www.sparkfun.com/products/12757
5. Martin, F. (2001). *Robotic Explorations: A Hands-On Introduction to Engineering*, Prentice-Hall.
6. Klassner, F. and Anderson, S., (2002). LEGO MindStorms: Not Just for K–12 Anymore. *IEEE Robotics and Automation, Special Edition on Education-I*, June 2003, v 9, no 2.
7. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, November 2010.
8. jEdit - Programmer's Text Editor. (2016). Retrieved from http://www.jedit.org
9. Budget Robotics (2016). *Build Your First Robot Chassis Kit (BYFR), From Servo Magazine - ARDBOT II*. Retrieved from the Budget Robotics, Inc. website http://www.budgetrobotics.com/
10. Optiboot. (2016). Retrieved from https://github.com/Optiboot/optiboot
11. Jamieson, P. (2010). Arduino for teaching embedded systems. Are computer scientists and engineering educators missing the boat?. *Proc. FECS*, 289-294.
12. Arduino UNO R3 (2013). Retrieved from https://www.arduino.cc/en/Main/ArduinoBoardUno

13. Dellaert, F., Fox, D., Burgard, W., & Thrun, S. (1999). Monte Carlo localization for mobile robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on* (Vol. 2, pp. 1322-1328). IEEE.
14. Lego (2016). *31313 MINDSTORMS EV3.* Retrieved from the Lego, Inc. website http://www.lego.com/en-us/mindstorms/products/31313-mindstorms-ev3
15. Hixon, R. (2007). Teaching Software Engineering Principles Using Robolab and Lego Mindstorms. *International Journal of Engineering Education*, October 2007, v23, i 5.