World Transactions on Engineering and Technology Education *© 2016 WIETE*
Vol.14, No.3, 2016

Teaching robot rapid prototyping for non-engineers - a minimalistic approach

Kimmo Karvinen

Aalto University
Espoo, Finland

ABSTRACT: Interdisciplinary involvement can bring out innovations and approaches to the robot design process that would not otherwise be possible. To utilise fully expertise from other areas than engineering, non-engineers need a basic robotics and embedded systems knowledge, allowing them to actively participate in robot development projects. In this article, a method of teaching robot rapid prototyping to non-engineers is proposed. A minimalistic skill set is defined, aiming to enable students to design and build basic robot rapid-prototypes. In addition to creating tangible results fast, the specified skill set also provides a solid context for elaborating different areas of embedded systems and robotics. Limited implementation of the method was tested in a two-day robot workshop in the University of Art and Design in Linz, Austria. The workshop was divided into one day of theory and one day of students building their robots. The case study indicates that the skills needed for designing and building basic robots and embedded systems can be summarised in a package that can be presented in one day.

INTRODUCTION

Traditionally research on teaching graduate and undergraduate robotics is focused on engineer education or various setups using robots in classroom, while learning goals are not on robot design. Other learning goals include, for example, teaching computer science [1][2], artificial intelligence [3] and programming [4].

As knowledge is scattered around the world and among people, even the best possible R&D organisation cannot be the expert in everything [5]. This leads to an assumption that having more interdisciplinary involvement from different areas of expertise in the robot prototyping process could bring out innovations and approaches that would not otherwise be possible. Non-engineers can be experts in their own area, but without some degree of robot and embedded systems knowledge they can only have a consultative role in robot development projects without being able to make prototypes or understand the technical aspects of the robot design process. On the other hand, engineers excel in technical aspects of the robotics, but have only limited knowledge of other areas of expertise. Increasing interdisciplinary robotics education allows non-engineer students to actively contribute to robotics R&D projects. However, the threshold of getting started with robotics can seem overwhelming for non-engineers as robotics is a mixture of both software and hardware with complex subareas, such as programming [6].

In this article, a method of teaching robot rapid prototyping to non-engineers is proposed with a focus on minimal skill set that is needed in order for students to design and build their own robot prototypes. The approach is minimalistic leaving out everything that is not absolutely necessary to reach the goal. Undoubtedly skill set attained this way cannot be very comprehensive, but it provides a solid context to elaborate different areas of embedded systems and robotics. Without clear application, separate areas may seem uninteresting and confusing for non-technically orientated students. In addition, quick tangible results create an inspiring and encouraging foundation for learning, which can be especially valuable when dealing with sociological barriers to programming and getting non-programmers interested about the programming part of the process [7].

Learning robot prototyping in two days was tested at a robot workshop at the University of Art and Design in Linz with 2nd and 3rd year students in the Department of Timebased and Interactive Media. Results with small group of nine students can not be generalised, but even so, it provides guidelines about how the method presented in this article works in practice. The workshop was split into two parts: one day of theory and one day of building a robot with simple behaviour. Observations of the course are based on perceptions during teaching, preliminary knowledge review forms and feedback/knowledge review forms afterwards.

This article is divided into four sections: method; rapid development platform; programming; and from *Hello, World!* to robot. The first section defines the features for a system that would qualify as a robot, specifying the desired outcome for the process. This is followed by presenting the learning workflow, which advances through sample setups to a defined outcome. Section two inspects the use and advantages of the rapid development platform, focusing on Arduino Uno. One of the main challenges of building embedded systems and utilising the rapid development platform is the requirement for certain degree of programming skills, which is examined in section three. The last section presents implementation of the sample setups that are needed in order to understand and build the defined outcome.

METHOD

A simple embedded system is a real-time system that consists of input, data processing and output: for example, a sensor, a microcontroller and a motor. Even the most complicated embedded systems follow the same structure, with more inputs, outputs and with advanced processing [8]. A robot is an embedded system with an actuated mechanism programmable in two or more axes and with degree of autonomy [9]. Many traditional robots are manipulators, used predominantly in industrial manufacturing, with their movements restricted to the reach of their arm [10]. Mobile robots, on the other hand, can move to fulfil their tasks and are mainly limited by their locomotion mechanisms [10]. In either case, at least one of the robot's outputs has to be a motor. Besides the ability to move and/or manipulate the environment, a robot has to be autonomous enough to complete its tasks based on sensor input without human interference [9]. These characteristics are presented in Figure 1.

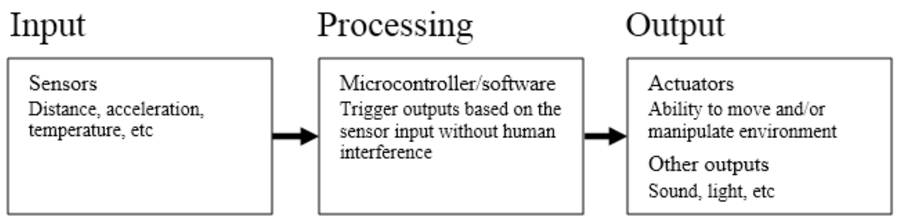

Input
Processing
Output
Sensors
Distance, acceleration, temperature, etc
Microcontroller/software
Trigger outputs based on the sensor input without human interference
Actuators
Ability to move and/or manipulate environment
Other outputs
Sound, light, etc

Figure 1: Input, data processing and output.

To teach students how to build and understand the basic principles of an embedded system with an input, data processing and output, the process is divided into smaller steps as shown in Figure 2. Each step is presented to students with ready-made sample code and a circuit diagram. The process gives students a basic understanding of the structure and the function of robots without focusing on individual sub-areas, such as programming, component properties or electric theory. After understanding how embedded systems work, students are able to prototype their own devices by using a suitable development platform and utilising available on-line community resources. The step approach also accustoms students to dealing with only one challenge at a time.

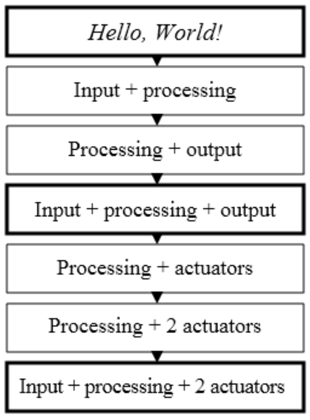


Figure 2: Sample setup steps.

The learning workflow starts from the simple *Hello, World!* program and advances through different sample setups, first to a basic embedded system and, finally, to an embedded system that qualifies as a robot as shown in Figure 2. Different steps introduce some of the more central embedded system programming concepts. Implementation of these steps is described in part from *Hello, World!* to robot. The required programming knowledge is not taught as a detached theory, but as a part of the sample setups, rendering the method very hands-on oriented.

RAPID DEVELOPMENT PLATFORM

Having a suitable development platform solves many low-level issues of prototyping. It connects inputs and outputs and provides an easy workflow for programming the data processing. With a development platform, students can build simple embedded systems without any significant understanding about electronics or without having specific mechanical skills, such as the ability to solder. Being *easy start* and producing instant results is valuable for lowering potential sociological barriers towards programming among non-engineers.

Only a small amount of electrical theory is needed in beginner prototyping. To be able to connect input and output components and to prevent breaking them, it is necessary to, at least, superficially understand concepts of voltage, current, resistance, open circuit, closed circuit, short circuit and the ground.

Many different prototyping boards are available, with Arduino Uno currently being the most popular one. In 2013, Arduino had registered over 700,000 official boards [11]. As Arduino is open-source, there is also an unknown number of non-official boards by other manufacturers in use. The advantages of using a board with a wide user base are emphasised in beginner use. A large on-line community provides support for problems and freely available code samples and tutorials. Connecting some components to a development platform can be challenging even for more advanced users, an issue, which can be solved by using a large selection of parts made by various manufactures, especially, for Arduino.

While Arduino has many interesting features for advanced users, beginners can build prototypes with a quite limited knowledge. Connecting to Arduino IDE, compiling, uploading and using a serial monitor are the most important basics skills. To be able to use components, it is necessary to understand the meaning of digital, analogue and power pins on Arduino. To avoid leaving these concepts on a theoretical level, students should try them out with sample setups, which are presented later in this article. When changing components between setups, the similarities between the different codes should be underlined to reinforce the idea of the equivalent basic structure.

While selecting a suitable development platform for teaching robot-prototyping for non-engineers it is essential to understand the strengths, as well as the limitations of the different boards. While Arduino is simple and straightforward to use, it is not capable of performing some more advanced tasks. For example, Raspberry Pi would be clearly a better choice for devices that need to send an email or analyse image. However, when beginners make their first prototype, it is reasonable to offer just one platform to begin with to provide an uncluttered and straightforward workflow.

For the Linz workshop, a simple rover robot platform was pre-made and assembled by the students. The platform was a chassis with two continuous servo motors attached to wheels and a caster wheel at the back. Arduino with external breadboard was fitted on the top. Rover robot allows students to combine easily different movement patterns to sensor input. The first student-designed robot prototypes based on this could be, for example, a maze solver, a line follower or a light seeking robot.

PROGRAMMING

Programming is one the most fundamental skills when designing robots and embedded systems. This may be one of the biggest challenges while teaching robotics, as students often regard programming courses difficult, which also shows in high drop-out rates [6]. Programming skills are formed from several overlapping domains and, there is a vast number of different things to learn and master [6]. It takes about ten years to turn a novice into an expert programmer [12]. Thus, when teaching people who are not aiming to be professional programmers, it is necessary to select, and start with, only the most central parts that are required for our goal of designing and building a simple robot. Table 1 shows parts of syntax used in the sample setups as will be presented in section from *Hello, World!* to robot.

Table 1: Syntax parts used in the sample setups.

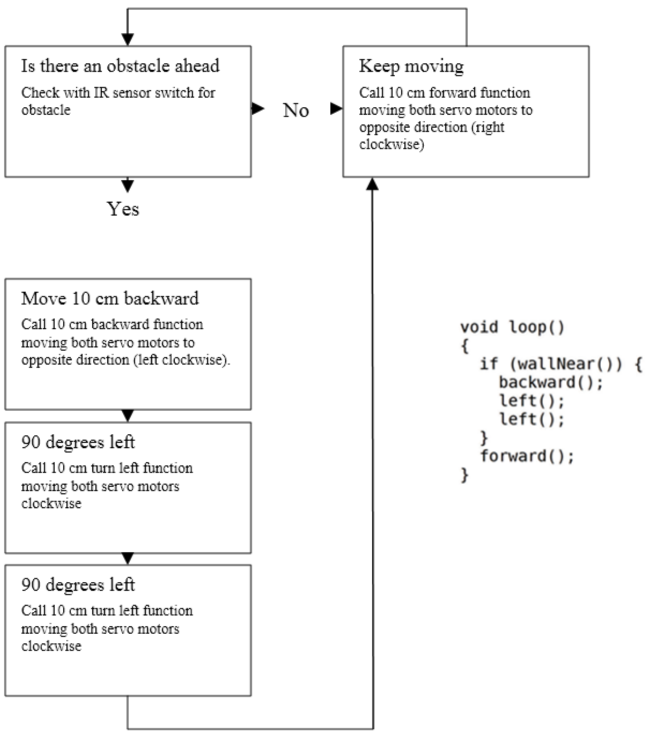
Hello, World! (blink)	IR sensor switch	LDR	Piezo speaker	Servo motor
int void digitalWrite delay pinMode	digitalRead bool else if Serial.begin Serial.print Serial.println	analogRead	while long int float delayMicroseconds	for

While using code examples and libraries lower the threshold, students have to be able to understand the structure and basic syntax of the code to be able to make their own programs, even if it is done by editing and altering existing code. Code examples made by manufacturers and the Arduino community allow building a wide range of embedded systems by making slight changes and combining different codes. Effectively, utilising those resources is the key skill for a beginner.

Programming knowledge and programming strategies can be separated from each other [6]. Programming knowledge, such as being able to state how *for loop* works, is the focus of most programming textbooks, as well as introductory programming courses [6]. However, for a non-engineer student building his/her first embedded system, programming is likely merely to be a tool to get the components working together. Hence, programming strategies driven teaching, focused on, for example, how *for loop* is applied and where is it needed, is probably more effective and interesting.

Control structures and syntax beyond sample setups were taught during the Linz workshop based on the needs of the students' own robot designs. For example, string comparing was needed in order to select which LDR senses more light.

Before students start writing any code after going through the sample setups, they should define the goals of the program and create a mental model of the program [6]. Mental models do not need to contain technical details, but the wanted behaviour has to be described in adequately small steps. When the objectives are defined, students can start finding solutions to separate steps to achieve the goal. One possible way to approach the program is to model behaviours from building blocks, as in Figure 3, then, program and test one block at the time.



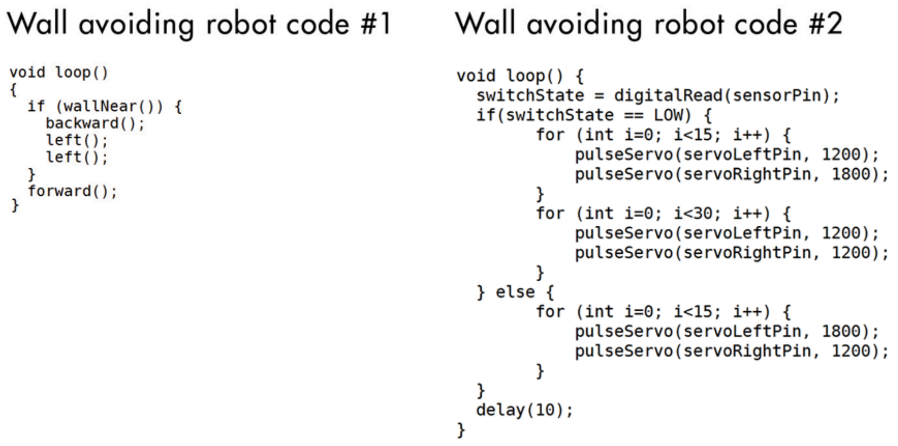
Is there an obstacle ahead
Check with IR sensor switch for obstacle
No
Keep moving
Call 10 cm forward function moving both servo motors to opposite direction (right clockwise)
Yes
Move 10 cm backward
Call 10 cm backward function moving both servo motors to opposite direction (left clockwise).
90 degrees left
Call 10 cm turn left function moving both servo motors clockwise
90 degrees left
Call 10 cm turn left function moving both servo motors clockwise

```
void loop()
{
  if (wallNear()) {
    backward();
    left();
    left();
  }
  forward();
}
```

Figure 3: Wall avoiding behaviour block diagram and code.

At the Linz workshop, movement of the rover robot was controlled by four functions: 90 degrees right, 90 degrees left, 10 cm forward and 10 cm backward. These can be easily combined into more complex movements. For example, the main program (main loop in Arduino code) is started by calling function checking for obstacle with IR-sensors. If an obstacle is detected, a 10 cm backward and two times 90 degrees left should be called for. If there is no obstacle, the main program keeps calling the 10 cm forward function indefinitely. Each block should be first tested individually in their own code and then individually as part of the main code. This kind of a process allows students to deal with only one challenge at the time, preventing them from getting overwhelmed by overlapping problems.

To keep the main program approachable and readable it can be written so that it reflects the building block structure. This can be achieved by using simple blocking conditional statements and function calls in the main program. As all more complex functionality is inside separate functions, the main program can remain semantically close to the mental model, which also makes trying out different behaviour patterns easy. Distinct programming styles can make a substantial difference as can be seen in Figure 4, presenting two ways of writing the main program for a wall avoiding robot. Selecting a simpler programming style can make a significant difference for students who are not accustomed to programming.



Wall avoiding robot code #1

```
void loop()
{
  if (wallNear()) {
    backward();
    left();
    left();
  }
  forward();
}
```

Wall avoiding robot code #2

```
void loop() {
  switchState = digitalRead(sensorPin);
  if(switchState == LOW) {
        for (int i=0; i<15; i++) {
            pulseServo(servoLeftPin, 1200);
            pulseServo(servoRightPin, 1800);
        }
        for (int i=0; i<30; i++) {
            pulseServo(servoLeftPin, 1200);
            pulseServo(servoRightPin, 1200);
        }
  } else {
        for (int i=0; i<15; i++) {
            pulseServo(servoLeftPin, 1800);
            pulseServo(servoRightPin, 1200);
        }
  }
  delay(10);
}
```

Figure 4: Two ways of writing main program for wall avoiding robot.

FROM *HELLO, WORLD!* TO ROBOT

This section presents one possible component selection for presenting sample setups. The selection can be varied depending on target group, prototyping goals and available components.

The *Hello, World!* program verifies that the development environment and the development platform are working. Arduino code, which blinks an on-board LED connected to pin 13, is generally used as *Hello, World!*. Blink code is also a good first example as it is short and very easy to understand, while still containing some of the most fundamental parts included in every code, such as running main program, changing pin state and declaring variable.

Using input can be sampled with any simple sensors. In order to show two very common sensor protocols, digital resistance and analogue resistance, for example, IR sensor switch and LDR (light-dependent resistor) can be used. The programs used introduce three important concepts: creating functions, conditional statement and Boolean function. As there is no output component attached at this point, the serial monitor is used to output values. Learning to use a serial monitor is essential as it enables students to test sensors individually without output components and also provides an efficient debugging tool.

A Piezo speaker is a suitable sample output. The Piezo speaker code presents a while loop, which repeats a block as long as the condition is true. For an novice, it may be easier to think of it as a repeating *if* statement. After testing it individually, it is combined with an IR sensor switch code with the *if* statement. Together these form a very simple embedded system that beeps when sensor is close to an object.

Hobby servo motors are well suited to use as beginner-friendly actuators. They have relatively high torque, are easy to mount and straightforward to control. To move the wheels, as in a rover robot, continuous servo motors are used instead of more common limited rotation models. The rapid changing of the digital output pin between *high* and *low* creates pulse, the length of which controls the speed and the direction of continuous servo motor. Even without understanding how square wave pulse works, students can use servo motors as long as they comprehend how the servo position is controlled by changing the pulse length value. The servo code also introduces a *for loop*, needed for running code for a number of iterations.

Moving two motors together by sending the pulse to another digital output pin is equally easy as moving just one. At this point, the robot platform can be assembled and basic directions for the robot programmed, resulting in a rover platform that loops between the four directions. In the final step, the already tested IR sensor switch is attached and the obstacle avoiding code described above is implemented.

Towards the end of day one at the Linz workshop, students were divided into groups of two and each group chose what kind of behaviour, input and outputs, they wanted for the robot. An assortment of about 40 sensors was provided along with the sample code to some of the sensors. All groups succeeded in building their robot during the second day. For example, the results included flame following and colour changing robots. According to feedback received, students found the workshop useful as the average rating for it was 4.4 on the scale of 1 to 5. Self-evaluations on their programming skills raised from an average of 2.3 to 2.9 (scale 1-5). Presumably, this tells more about how complicated and approachable students see programming after the course than the change in the actual skill level.

DISCUSSION

The presented workflow has some similarities to courses using Lego *Mindstorms* and similar easy accessibility programming environments. Using Arduino solves some of their main drawbacks, but brings out different kinds of challenges. Arduino code is close to C++ and it is written in the traditional fashion, teaching students skills that can be applied to all major programming environments. *Mindstorms* or similar systems using a GUI-based programming environment do not have this advantage, although it is fair to mention that *Mindstorms* have a work-around for this by using NQC and RCX-Ada [3]. Closed systems, such as *Mindstorms* also limit the range of possible components, narrowing its use for innovative prototyping [3]. In contrast, Arduino is capable of operating with all kinds of inputs and outputs from various manufacturers.

Probably the main advantage of using Arduino for prototyping instead of dedicated learning environments is that Arduino can also be used for serious prototyping [13]. This gives students the option to use learned skills for commercial and industrial prototypes. As Arduino Uno uses an Atmega328 microcontroller, found also in many commercial products, in some cases, the code made for the prototype can be used directly in the final product. Naturally, these advantages do not exclusively concern Arduino, but are relevant to any similar type of development environment with equivalent characteristics.

One of the major challenges of the presented approach is the fact that students need to be able to keep the syntax intact. Despite the efforts to make programming languages simple and understandable, many beginners struggle with the syntax [7]. This was also the case at the Linz workshop and it was the area where continuous support was needed. To avoid frustration at this point, debugging should be made into a natural part of the programming process from the beginning. Building in small steps, dealing with one problem at the time and using code examples helps to isolate the problem. Instead of solving syntax problems for the students, advice with debugging was provided. The students found it satisfying to notice that they could fix the broken code and take control of a confusing situation.

Lego *Mindstorms* and similar kits have one clear advantage that supports their use: they provide a quick and simple way to build various mechanical structures. Building even an exceedingly simple platform, such as the rover platform, takes time and requires the use of power tools.

Even though the bigger picture of student prototyping and outcomes is outside the scope of this article, there is one notion that rises from various skills needed. It is not likely that most non-engineers learning embedded systems would become experts on technical aspects. As stated before, mastering just one part of the process, such as programming, takes a lot of time. However, students are learning to be experts in their own field, which is the feature that should be combined with embedded systems and robotics in order to bring out significant outcomes.

CONCLUSIONS

This experimental case study indicates that the skills needed for designing and building very basic robots and embedded systems can be summarised in a package that can be presented in one day. Arduino Uno is one suitable option for a development platform. The most significant challenge for non-engineers is the need for certain level of programming skill. Getting started with the programming of embedded systems can be eased by limiting used syntax only to necessary terms. Creating a mental model of the program and splitting it in manageable pieces is vital in order to deal with limited amount of challenges at the time. Programming style can make a substantial difference to how approachable and how close to the created mental model the program is. These aspects combined provide a minimalistic teaching approach, lowering barriers for non-engineers engaging in robot prototyping projects and supporting utilisation of interdisciplinary expertise.

At the Linz workshop, every group successfully built their robot in the given time (one day). However, students' robots were based on a pre-built robot platform. Without it, the time frame would have been inadequate. It would be interesting to test what students could do in a longer workshop with more time for designing and building their own robots.

ACKNOWLEDGEMENT

The author wishes to express his thanks to Tero Karvinen for comments on the method and sample programs, and to Ville Kyrki for comments on the manuscript.

REFERENCES

1. Fagin, B.S., Merkle, L.D. and Eggers, T.W., Teaching computer science with robotics using Ada/Mindstorms 2.0. *ACM SIGAda Ada Letters*, 21, **4**, 73-78 (2001).
2. Fagin, B. and Merkle, L., Measuring the effectiveness of robots in teaching computer science. *ACM SIGCSE Bulletin*, 35, **1**, 307-311 (2003).
3. Klassner, F., A case study of LEGO Mindstorms TM suitability for artificial intelligence and robotics courses at the college level. *ACM SIGCSE Bulletin*, 34, **1**, 8-12 (2002).
4. Lawhead, P.B., A road map for teaching introductory programming using LEGO$^{©}$ mindstorms robots. *ACM SIGCSE Bulletin*, 35, **2**, 191-201 (2003).
5. Chesbrough, H., *Open Innovation: Researching a New Paradigm*. New York: Oxford University Press, 2 (2008).
6. Robins, A., Learning and teaching programming: a review and discussion. *Computer Science Educ.*, 13, **2**, 137-149 (2003).
7. Kelleher, C. and Pausch, R., Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37, **2**, 90-132 (2005).
8. Laplante, P. and Ovaska, S., *Real Time System Design and Analysis*. USA: Wiley-IEEE Press, 3-4 (2011).
9. ISO 8373:2012 (en), Robots and Robotic Devices - Vocabulary, 29 October 2014, https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en
10. Siegwart, R. and Nourbakhsh, I.R., *Introduction to Autonomous Mobile Robots*. Cambridge, Massachusetts: The MIT Press, 1-2 (2011).
11. MEDEA, Arduino FAQ - with David Cuartielles (2013), 29 October 2014, http://medea.mah.se/2013/04/arduino-faq/
12. Winslow, L.E., Programming pedagogy - a psychological overview. *ACM SIGCSE Bulletin*, 28, **3**, 18 (1996).
13. Karvinen, K., Tikka, T. and Praks, J., Using hobby prototyping boards and commercial-off-the-shelf (COTS) components for developing low-cost, fast-delivery satellite subsystems. *J. of Small Satellites*, 4, **1**, 312 (2015).