Hedgehog: A Versatile Robotics Platform for Education and Advanced Applications
Timon Höbert, Reinhard Grabler, Clemens Koza, Christoph Krofitsch
Practical Robotics Institute Austria, Email: [hoebert, grabler, koza, krofitsch]@pria.at

Hedgehog: A Versatile Robotics Platform for Education and Advanced Applications

1 Introduction

Unfortunately there has been a distinct lack of student interest combined with a lack of adequately trained workforce in the domains of science, technology, engineering and math (STEM) in the recent years. Therefore, it is more important than ever to introduce students to STEM [1], [2]. Robotics is a suitable instrument for teaching purposes in STEM on various educational levels due to its interdisciplinary nature and its appeal to young people [1], [2], [3], [4]. Early exposure to these fields increases the impact of taken efforts.

Young people are fascinated by autonomously acting machines such as robots. This fascination and the variety of fields and topics covered make robotics a powerful idea to engage with, which requires teamwork, creativity and entrepreneurial skills for the design, programming and innovative exploitation of its possibilities. Creating and programming a robot represents an interdisciplinary project and involves topics such as movement, navigation, coordination, grasping, audio and video processing, as well as cognition and recognition. Thus, it supports the development of systems thinking, problem solving, self-control, and teamwork skills [5], [6].

However, school budgets are often too small for introducing robotics at young ages due to the required investments in equipment. Furthermore, teachers trained adequately in robotics are often not available for these ages. As a consequence, only individual teachers efforts dominate robotics education [1], [2], [7]. Despite the fact that the robotics industry provides pre-programmed prefabricated robots likewise to black box solutions for the industrial environment, the educational domain rather involves white box solutions, which aims at building robots from scratch [8]. For leveraging such solutions to a broader application in formal education, the programming technologies should be easy to use and understandable for allowing also teachers without extensive computer science backgrounds to use them in their lessons. Generally, it would be desirable to reduce the programming requirements for robots accordingly [9]. This led to the development of Hedgehog [10], a low priced robot controller that involves smartphones and similar mobile devices. Smartphones provide a rich user interface via touch screen, network capabilities, fast processors, a lot of memory, and internal sensors, such as gyroscopes, acceleration sensors, and cameras [10]. With smartphones getting increasingly common among even young school children, duplicating these capabilities in a robot controller unnecessarily increases costs. At the same time, using the smartphone further engages students in the subject matter. Supplementary material and intuitive user interfaces support teachers without specific training and aid teachers and children in a shared learning and exploration experience.

Using mobile devices lets Hedgehog automatically benefit from innovations in the field of consumer electronics. Network capabilities and other features less important for introductory robotics courses open up a wide area of applications in and beyond the education domain [11], [12]. These benefits make Hedgehog less susceptible to obsolescence and thus a versatile long term investment.

2 Concept

Figure 1 shows the Hedgehog Architecture. It is designed to be used in different configurations depending on the users needs and capabilities.
The first configuration uses the controller as programmable black box. User programs can be written and deployed over WiFi via the Hedgehog desktop IDE or via the Hedgehog smartphone app. To do so, the IDE offers a number of library function headers to use, e.g. for reading sensor values or driving motors. When deploying, the Software Controller (SWC) receives user programs as source code, compiles them directly and saves them. During the execution the concrete hardware commands are transmitted to the Hardware Controller (HWC).
The Hedgehog app also provides the whole graphical User-Interface (GUI) for the controller to test connected motors and sensors. This GUI eliminates the need for an onboard screen found on most conventional educational controllers.
The HWC is responsible for hardware specific tasks and is effectively a black box to the user. These tasks include generating pulse-width modulated signals for motors and servos, providing WiFi connectivity, or communicating with external hardware. Because of the easy usage of this configuration, it is designed for students to learn the basics about robotics.

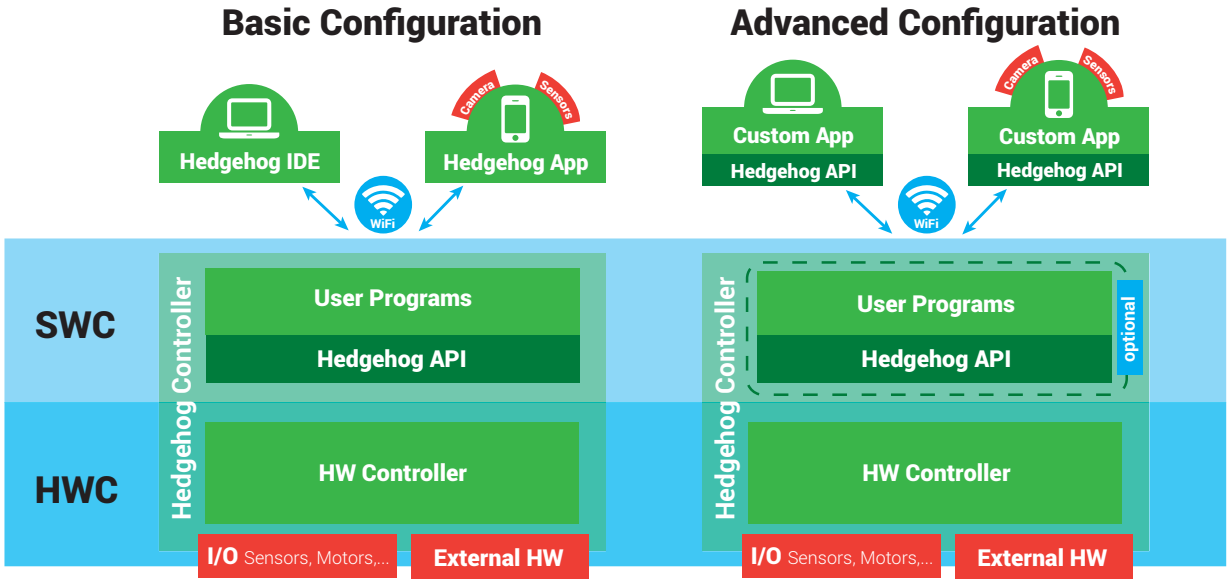

Figure 1: Hedgehog architecture showing the core components and interfaces [10].

The second configuration shows a possible use for advanced applications, where the user can program different components on different layers. On the top layer custom apps use the Hedgehog

API to provide a custom graphical User Interface and application logic. This API is used to communicate to the HWC and the user programs on the SWC. The usage of the SWC is not necessary but recommended. By encapsulating the hardware specific functions as a User program, a flexible system architecture can be achieved. The direct connection between SWC and HWC is especially beneficial for fast and high precision hardware-specific functionalities.

2.1 Communication Interfaces

All communication in Hedgehog is based on stateless message passing protocols incorporating flow control and error containment. The Hedgehog Command Protocol (HCP) defines commands for interfacing the HWC, e.g. for reading sensor values. For program management, the Hedgehog Deployment Protocol (HDP) is used, which defines commands for compilation, compilation result, execution, and execution states. Furthermore, commands are incorporated for transferring source code from the SWC to the application, which enables a newly connected smartphone to import all user programs and work with them as if they have been developed on it.
The applications communicate with the Controller via WiFi, which is realised through a WiFi module that is integrated into the HWC. The HWC and SWC communicate via a wired serial interface. Deployment commands between the app and the SWC are simply being forwarded through the HWC. This architecture allows the SWC to be omitted in case no robot programming is desired, while the app still has access to the robot hardware.

3 Implementation

In its current stage, the Hedgehog apps IDE supports programming in C. Support for other popular languages, such as Python or Java, is planned for the future. To accommodate for the heavy use of punctuation in most programming languages, additional code template buttons, e.g. for inserting loops, are provided in the IDE. During deployment, the source code is sent to the SWC and compiled there. Compilation results, i.e. return code and any compiler messages, are sent back to the application. The alternative, compiling on the application layer, would require cross-compilation support. This is not viable on most mobile devices, and would introduce tighter coupling between application and SWC, which is not desirable. For debugging, commands such as Debugging Break Action, which suspends the running SWC program, are available to the application. These commands correspond to commands and events of Gnu Debugger (GDB), although the use of GDB on the SWC is an implementation detail. Figure 2 shows two screenshots from a Hedgehog debug session.

3.1 Integrated Development Environment

For the IDE development we focused on platform-independent concepts that can be easily used or adapted to different platforms. The platform-independent code of Hedgehog was developed in a separate Java project containing the protocol implementations, user program management and easy-to-use wrapper classes. This project serves as library or guideline for different platform implementations. We already implemented the desktop IDE, the app for Android devices and a version for iOS is currently in development.

The user experience should be the same on different platforms while taking advantage of the specific graphical user interface (GUI) concepts. The IDE interface will be separated in a screen for managing programs, a screen for managing the versions of a program and a screen for editing a program. The layout of the latter is shown in Figure 2.

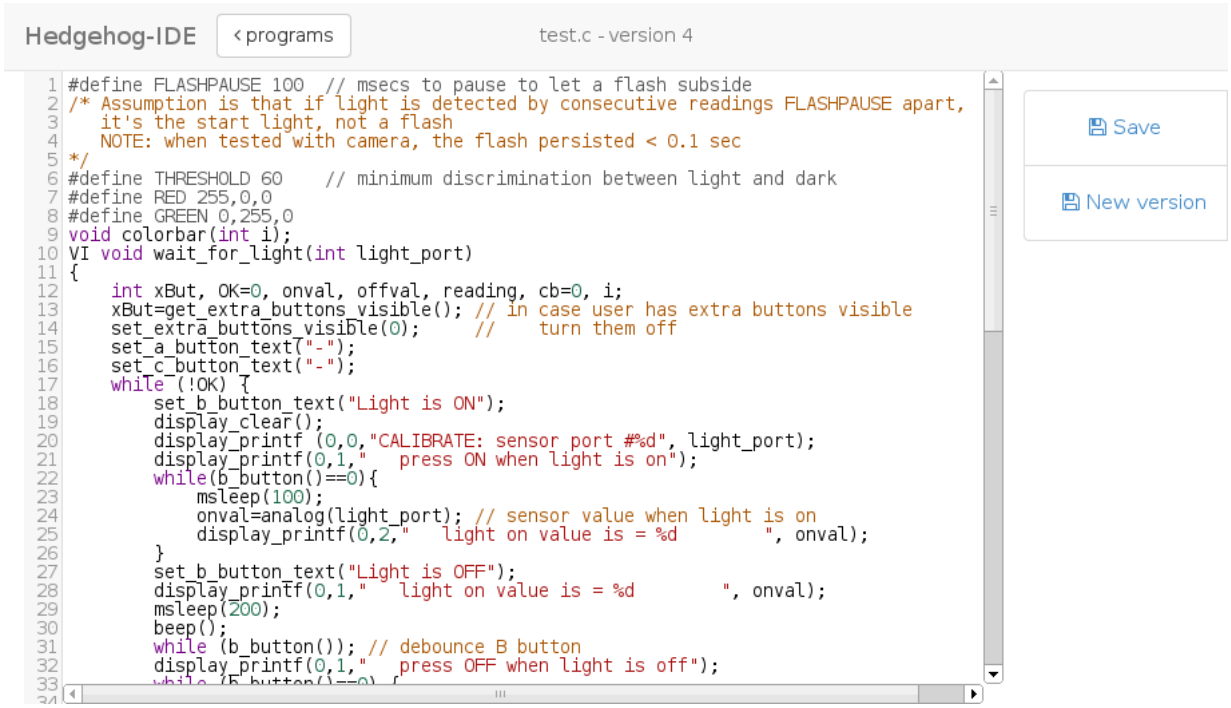

Figure 2: Editor screen layout of the desktop IDE. With the buttons on the right, the program can be saved and versioned.

3.2 Software Controller

As Software controller we are using the Raspberry Pi Model A, as it is inexpensive and easy to use. The software on the Raspberry Pi is written in C and encompasses all user program management functionality on the SWC side. This includes implementing HDP, program reception, compilation, execution, saving and memory management. Currently, the SWC supports user program management for C programs, but in future this will be extended to other programming languages as well. The Raspberry Pi software is mainly structured as consumer for incoming commands; user programs will be executed in child processes. Figure 3 depicts the control flow.
After initialization, the software waits for an incoming command from either the app, the HWC or possible child processes. Generally, a received command is executed if the SWC supports it. The following HDP commands are pointed out as they have a significant role in the user program management:

- When receiving a *compile command*, include statements will be added to the source code in order to link the right library functions for the currently connected HWC type. After saving the source code, the program will be compiled and linked and the results will be sent back to the app.

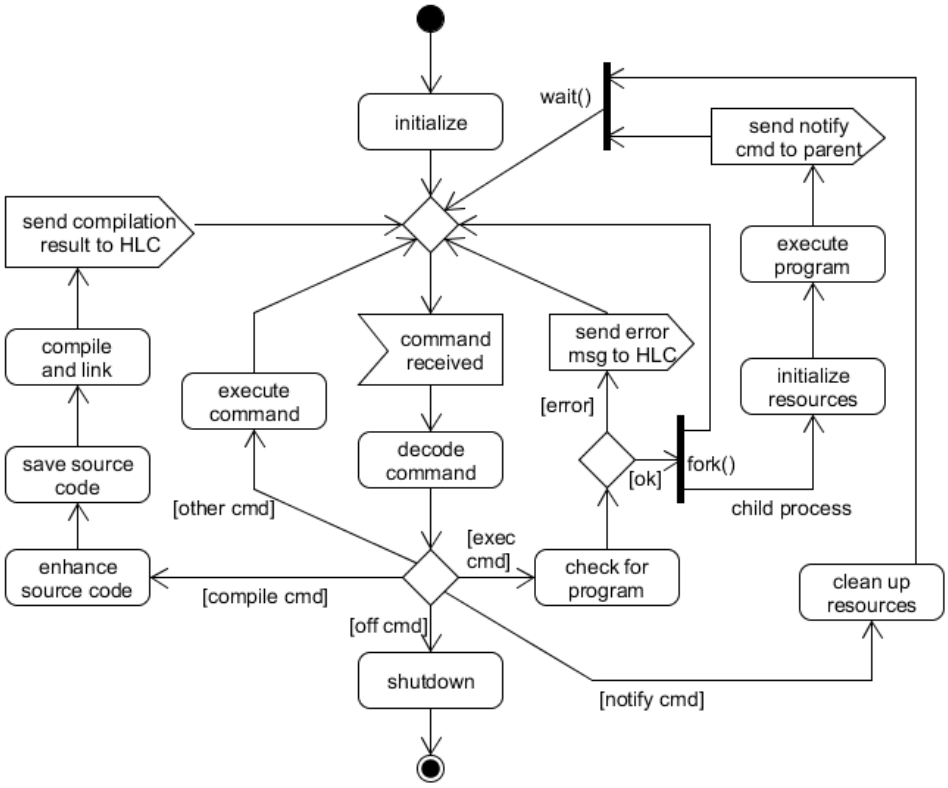

Figure 3: Activity diagram showing the most significant elements of the control flow of the software running on the SWC.

- When receiving an *execute command* for an existing program, a new child process will be created, running simultaneously in the SWC software representing the parent process. The child process then executes the main-function of the user program and after returning, it informs the parent process using a notify command.
- When receiving a *notify command*, resources will be cleaned and the child process properly terminated.
- When receiving a *fetch command*, all saved programs including all versions will be transferred to the app (not specifically shown in Figure 3).

While a child process is executing a user program, it will eventually need to interface the robot hardware, e.g. for reading sensors or powering motors. As the SWC software (parent process) manages the interface to the HWC, requests from the user program need to be forwarded. Therefore, a pipe connection is established to each child process the software creates, keeping track of its forwarded requests and passing the answers back accordingly. Figure 4 illustrates this concept with an example user program.
User programs also have the opportunity to receive and send custom data from and to the application, which is particularly important in more complex applications with a custom high-level program. The commands for that are also forwarded in the way that is shown in Figure 4.

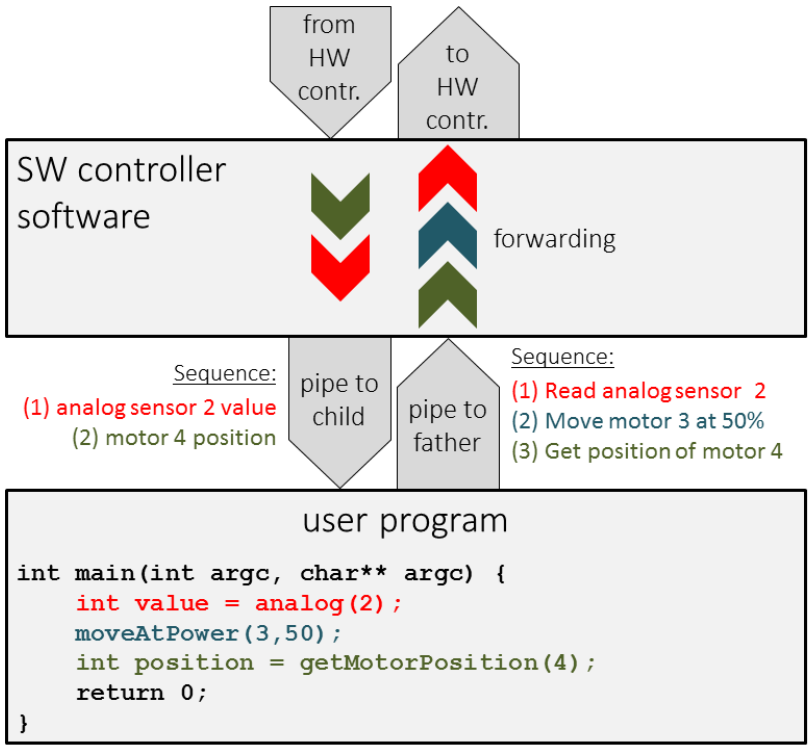

Figure 4: When a user program is executing, its commands to the HWC and vice versa are forwarded accordingly. In this example, the program first tries to read the value of an analog sensor. The analog sensor value request is forwarded to the HWC, and the value in the according answer is then assigned to the variable "value". Since powering a motor is asynchronous, there is no answer for that command. Reading the position of a motor works similarly to reading the analog sensor value.

3.3 Hardware Controller

The HWC enables to connect and control sensors, motors and servos, handles time critical functionality like Pulse-width modulation (PWM) generation and motor speed measurement, parses and manages the communication with the application and SWC and powers the Controller via battery. For the main processing unit of the HWC an ARM-Cortex M4 based STM32F3 series processor with a clock speed of 72 MHz was chosen. Its rich 100 pin package together with two additional shield PCBs (printed circuit boards) allows connecting and controlling 16 digital sensors, 16 analog sensors, 6 motors and 6 servos. For communication the STM32 offers a rich number of interfaces, whereof two 3.3V UART interfaces are used for the WiFi module and SWC (Raspberry Pi). Other UART, SPI and I^2C interfaces are free for possible extensions in future. For powering the whole Controller, a 9.6V 2100mAh NiMH battery pack is used, including a charge controller which keeps track of charging and discharging cycles and therefore the current battery state. In general, focus was laid on only using affordable components in order to keep the Controller at a low cost and thus aim for a broad educational application. Figure 5 shows the current prototype of the HWC.

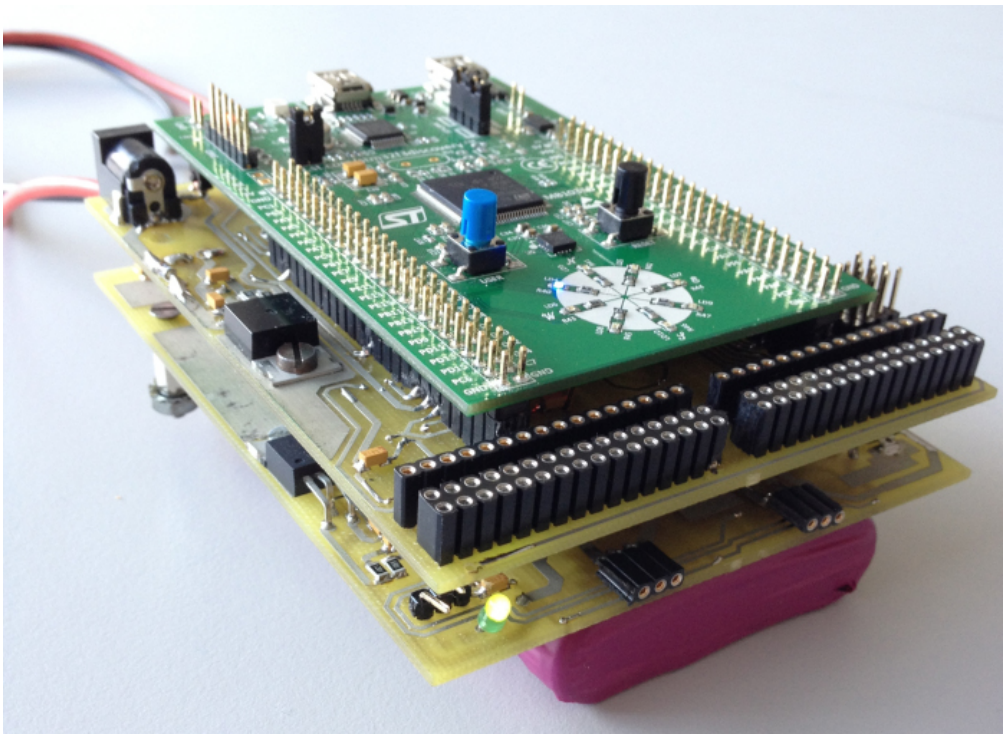

Figure 5: Current Hedgehog HWC on top of its battery.

4 Conclusion and Future Work

The Hedgehog controller represents a powerful platform for educational robotics. Using smartphones facilitates existing resources and provides a well-understood, attractive working environment. At the same time, Hedgehog can leverage future improvements in those devices. Overall, this makes Hedgehog a worthwhile low-cost, long-term investment for schools. As mentioned before, iOS support and debugging features are currently in development. Additional programming languages will make Hedgehog attractive to a broader audience. Graphical programming for the Hedgehog platform, based on Catrobats Pocket Code [13], is already in its planning stages and will make Hedgehog suitable for young children.

Acknowledgment

The authors would like to acknowledge the financial support of the COIN program, an initiative by the Austrian Federal Ministry of Science, Research and Economy as well as Austrian Federal Ministry for Transport, Innovation and Technology. Thanks to Christoph Krofitsch, Clemens Koza and all other students who were involved in the development of the platform in recent years, for their passion to make this project possible.

References

[1] M. J. Mataric, N. P. Koenig, and D. Feil-Seifer, "Materials for enabling hands-on robotics and stem education." in *Proceedings of the AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education,*, Stanford, CA, 2007.

[2] G. Koppensteiner, M. Merdan, and D. P. Miller, "Teaching botball and researching disbotics," *Robotics in Education*, 2011.

[3] D. Alimisis, J. Arlegui, N. Fava, S. Frangou, S. Ionita, E. Menegatti, S. Monfalcon, M. Moro, K. Papanikolaou, and A. Pina, "Introducing robotics to teachers and schools: experiences from the terecop project," *Proceedings for Constructionism, American Univ. of Paris*, vol. 1, pp. 1–10, 2010.

[4] A. Bredenfeld, A. Hofmann, and G. Steinbauer, "Robotics in education initiatives in europe-status, shortcomings and open questions," in *Proceedings of Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2010) Workshops*, darmstadt, 2010, pp. 568–574.

[5] J. Johnson, "Children, robotics, and education," *Artificial Life and Robotics*, vol. 7, no. 1-2, pp. 16–21, 2003.

[6] I. M. Verner and D. J. Ahlgren, "Robot contest as a laboratory for experiential engineering education," *J. Educ. Resour. Comput.*, vol. 4, no. 2, Jun. 2004.

[7] D. Alimisis, "Integrating robotics in science and technology teacher training curriculum," in *Proc. Int. Workshop Teaching Robot. Teaching Robot., Integr. Robot. School Curric*, 2012.

[8] C. Kynigos, "Black-and-white-box perspectives to distributed control and constructionism in learning with robotics," in *Workshop Proceedings of SIMPAR*, 2008, pp. 1–9.

[9] B. Lazinica, A. Katalinic, "Self-organizing multi-robot assembly system," *INTERNATIONAL SYMPOSIUM ON ROBOTICS*, vol. 36, p. 42, 2005.

[10] C. Krofitsch, C. Hinger, M. Merdan, and G. Koppensteiner, "Smartphone driven control of robots for education and research," in *Robotics, Biomimetics, and Intelligent Computational Systems (ROBIONETICS), 2013 IEEE International Conference on*, Nov 2013, pp. 148–154.

[11] S. Abbas, S. Hassan, and J. Yun, "Augmented reality based teaching pendant for industrial robot," in *Control, Automation and Systems (ICCAS), 2012 12th International Conference on*, 2012, pp. 2210–2213.

[12] R. Aroca, L. Gonçalves, and P. Oliveira, "Towards smarter robots with smartphones," *Robocontrol 2012*, 2012.

[13] Catrobat, "Pocket Code (website)," URL: http://www.catrobat.org, online; accessed 2015-05-26.