Architectural Overview and Hedgehog in Use

Clemens Koza, Martin Wolff, Daniel Frank, Wilfried Lepuschitz, and Gottfried Koppensteiner

Practical Robotics Institute Austria (PRIA), Vienna, Austria,
{koza,wolff,frank,lepuschitz,koppensteiner}@pria.at,
WWW home page: https://pria.at/team/

Abstract. Robotics is a versatile tool for teaching STEM topics, as it supports various disciplines, skill sets and target audiences. However, controllers used in Educational Robotics are often limited in their use cases. In this regard, Hedgehog tries to be flexible by design. This paper introduces Hedgehog’s architecture, currently implemented and future use cases, and experiences from our first Hedgehog workshops.

Keywords: Raspberry Pi, controller, visual programming, Blockly, Python

1 Introduction

As a joint research and educational organization, the Practical Robotics Institute Austria (PRIA) offers students various ways to participate in educational robotics activities. From elementary school, where PRIA conducts LEGO Mindstorms courses, to technical high school, where students participate in international robotics competitions and research projects, there are a lot of different approaches and requirements to our offers.

As such, PRIA uses a variety of robotics controllers depending on the use case. However, investing in and supporting different systems results in more complex maintenance and higher costs. A device that offers a wide variety of applications for audiences of different ages, experience levels and interests is therefore desirable. With Hedgehog, we have striven to create such a system.

2 Related Works

There are many different robot controllers on the market. Here, we show three controllers in use at the PRIA lab: LEGO EV3, “Link,” and “Wallaby”, the latter two by KIPR, an American organization that runs the Botball Program.

The LEGO Mindstorms EV3 [1] is a robust, commercial robot controller. It uses a visual programming environment for coding, programs are transferred to the controller over a Mini USB cable. While the commercial support and the thought-out accompanying robotics kit are a big benefit, it is also a closed system. Up to four motors and sensors each are connected to the EV3 using proprietary cables. Servos are not supported, and the focus is clearly on children, with no official way to do textual programming on the EV3.
The KIPR Link and Wallaby controllers [2] were developed for the Botball [3] robotics competition. They are both programmed in C, and thus not suitable for elementary school children. The Link uses native desktop IDEs (Integrated Development Environments) for Windows and OS X, and WiFi or USB for transmission. The USB driver is known to make problems in some situations.

The Wallaby is connected to a development machine via WiFi or USB, where it is detected as a network device. In some Windows configurations, driver installation has problems, but as long as the driver is installed properly, it works reliably. By default, each Wallaby opens a wireless hotspot with a unique SSID. In situations where many Wallabies are used at the same time, wireless connections don’t work any more. Consequently, a software update was released that gives the user the option to deactivate the WiFi, or connect to another network.

A great obstacle for using one of these controllers for different audiences is their programming languages. The EV3 was not developed to get an update for textual programming, and Link and Wallaby were not developed for visual programming. Other than that, none of these controllers support the user in combining the robot API (Application Programming Interface) and other libraries.

3 Design & architecture

Hedgehog’s versatility stems to a large extent from its architecture. Furthermore, Hedgehog embraces open source technologies, enabling and inviting users to tinker with almost all parts of the controller.

The controller consists of two main parts: the Software Controller (SWC) is a Raspberry Pi 3 Model B that provides connectivity via WiFi, Ethernet, and USB. It is master to the Hardware Controller (HWC), which handles external components. The HWC is implemented as a printed circuit board (PCB) with an STM32F4 microcontroller at its core. [4]

A controller is accessed over the network using WiFi or Ethernet, or from the device itself. Programs can be written using a web-based IDE.

The following subsections provide a bottom-up view of Hedgehog’s hardware and software architecture. While many users will mostly use the Hedgehog IDE to write programs, all of these parts are open to examination.

3.1 Hardware Controller (HWC)

One of the Hardware Controller’s main purposes is providing the physical connection to peripheral hardware. Table 1 gives an overview of the connectors, some of which are routed through from the Raspberry Pi. The pins are compatible to the Link and Wallaby controllers.

Hedgehog uses an external 7.4V power source, such as a 2-cell Lithium Polymer battery. For power management, the HWC board contains two switching regulators: one for the Raspberry Pi, and one for the servo ports. The motor drivers are connected directly to the battery. The HWC contains a buzzer
| Connector   | Count | Description                                                                 |
|------------|-------|-----------------------------------------------------------------------------|
| Motor      | 4     | 2 pin connector, motor driver                                               |
| Servo      | 4     | standard RC servo connectors                                                |
| Digital I/O| 8     | 3 pin connector, configurable as input & output; optional pullup or pulldown resistors for input |
| Analog Input| 8    | 3 pin connector, 12-bit analog/digital converter inputs with optional pullup or pulldown resistors |
| I²C Bus    | 1     | Inter-integrated circuit from Raspberry Pi                                   |
| SPI        | 1     | Serial Peripheral Interface from Raspberry Pi                               |
| SPI        | 2     | Serial Peripheral Interface from Microcontroller                            |

Table 1. HWC connectors overview

and four programmable LEDs, and monitors the battery voltage. Currently, the buzzer is used to issue a low-voltage warning.

HWC and SWC interface via Universal Asynchronous Receiver/Transmitter (UART) [4]. In addition, the microcontroller’s reset and boot pins are connected to the Raspberry Pi, allowing replacement of the firmware using the STM32F4’s UART bootloader.

The default firmware services requests by the SWC, but future versions might use HWC-initiated communication as well, e.g. for interrupt-based sensor input.

3.2 Software Controller (SWC)

The Software Controller contains most of the Hedgehog’s logic. It uses Raspbian and runs the Hedgehog’s server processes.

Hedgehog’s main process is a python program called the “Hedgehog Server”. It services commands that are received over the network, communicates with the HWC, and provides a discovery service so that Hedgehog controllers can be found on the network. The server is modular to simplify updates and customization, e.g. addition of new message types. The required software can be installed using the Python package manager, pip, so that software updates are easily possible.

The SWC also runs the “Hedgehog IDE” web application, which will be available via the node package manager, npm. Furthermore, an STM32F4 toolchain [4] and the Protobuf compiler are installed on the Raspberry Pi, meaning all Hedgehog software can be built and deployed using the controller.

The installation process is currently best documented for Unix-like operating systems. It is a clear and easy procedure starting with an empty Micro SD card. Easy installation encourages development and testing of modified software setups. For example, the default text-only Raspbian can be replaced with one providing a graphical shell, otherwise following the same installation instructions.

As Hedgehog does not have a touch screen or similar, a network connection is required for using it. This leads to the problem of a sensible default network configuration. Hedgehog solves the problem by using a configuration file provided
on a USB flash drive. At startup, the controller looks for this file and saves the settings, including networks and the controller’s device name.

By default, Hedgehog does not open a WiFi hotspot, it instead connects to an existing network. The rationale is that in a classroom setting, having one wireless network per controller results in terrible performance. Ad-hoc networks are not universally supported and were therefore not considered further.

3.3 User Programs & Hedgehog Protocol

By accessing the controller over the network, programs can be run on any machine. In the usual case, a local process connects to the Hedgehog Server, but that is not required. For example, one program could connect to multiple controllers at the same time, as shown in Listing 1.

```python
from time import sleep
from hedgehog.client import connect

# connect to two Hedgehog controllers. Default is to connect
# to the local Hedgehog Server on 127.0.0.1:10789.
# Specifying 'endpoint=None' overrides this to use the
# discovery mechanism: only a controller that is configured
# as a 'robot-arm' is considered for connection.
with connect() as hedgehog1,
    connect(endpoint=None, service="robot-arm") as hedgehog2:
    # move local motor for one second
    hedgehog1.move(0, 1000)
    sleep(1)
    hedgehog1.move(0, 0)

    # enable and move remote servo
    hedgehog2.set_servo(0, True, 1500)
    sleep(0.5)

# upon termination, all motors and servos are disabled
```

Listing 1: Python sample program connecting to multiple controllers

The Hedgehog Server listens to a well-known port, so local programs can use that port and the device’s loopback address to connect to it. For remote applications, users can either specify the IP address or device name manually, or use the discovery mechanism to find controllers that advertise a particular service. For example, a program might connect only to a controller that offers the custom `robot-arm` service.

Connections between client processes and the Hedgehog Server use ZeroMQ [5], a messaging library. Protocol messages are encoded and decoded with Protobuf [6]. Both of these libraries are available in many programming languages,
allowing for client programs in various languages [4]. Client libraries are intended to follow the philosophy of the language in question so that using Hedgehog, that language can be taught properly.

The Hedgehog protocol and client libraries support accessing sensors, motors, and servos. In addition the protocol defines messages to start and manage processes on the controller, e.g. to run user programs from an IDE. Protobuf allows it to handle unknown and changed message types, enabling updates to the protocol while maintaining compatibility on the wire.

An important design goal of using a language-agnostic, extensible network protocol for all hardware access is the ease of implementing new kinds of clients. For example, a first prototype of the Hedgehog IDE was a standalone Python desktop application. The new web-based version is a very different kind of software, but it can access the controller in the same way.

An example program using the Python client is shown in Listing 1. There is also a node.js client library, and a Scala/Java library prototype for Android apps, which was used to create a mobile visual programming environment [4].

### 3.4 Hedgehog IDE

The Hedgehog IDE is a web application that uses node.js for the backend and Angular 2 for the frontend. Both are written in TypeScript and share code where possible. From a Hedgehog architecture point of view, it is a rather complex, but otherwise ordinary client program: it runs locally to the controller and connects to the Hedgehog Server using the node.js client library to run user programs.

![Hedgehog IDE Python editor](https://example.com/hedgehogIDE.png)

**Fig. 1.** Hedgehog IDE Python editor.
The Hedgehog IDE allows users to create, edit and execute applications for the controller. Internally, all projects are managed as git repositories. This is normally hidden from users, but it creates a universal way for advanced users and third party tools to access Hedgehog projects.

The IDE supports both textual and visual programming, shown in figures 1 and 2, respectively. The latter is based on Blockly, a library for building visual programming editors. The visual editor provides two “toolboxes” of command blocks, one targeted at beginners, and a more comprehensive toolbox for advanced users. Blockly programs translate directly to Python code that is then executed. Users can inspect the code that is created from their visual program and thus use the visual editor to transition to textual coding.

![Blockly editor](image)

**Fig. 2.** Hedgehog IDE Blockly editor.

As a web application, the IDE doesn’t need to be installed on the programming device, and supporting different platforms does not require separate applications. Access via WiFi or Ethernet does not require any driver installation, and no firewall rules are necessary to access the IDE. Although development is currently focused on traditional desktop computers and browsers, the IDE can later be optimized for mobile devices.

When using a Raspbian version with a graphical shell, Hedgehog IDE can be used directly on the controller by connecting peripherals to the Raspberry Pi’s HDMI and USB ports.
Most parts of Hedgehog can be assembled using equipment typically present in fab labs. In particular, a laser cutter for the acrylic case of the controller and a reflow oven for SMD (surface mounted device) assembly are required. [4]

For space reasons, the HWC circuit board has four layers, making it one part that can not easily be manufactured in fab labs. Other hardware components that need to be purchased include a battery, Raspberry Pi, SMD parts and pin headers, and acrylic sheets and screws for the case. [4]

Hedgehog runs mostly free, open source software: Raspbian is a variant of the open source Debian Linux distribution. All parts of the Hedgehog Server, IDE, and firmware are licensed under AGPLv3 and available from GitHub. Required third party software, such as ZeroMQ, are open source and installed from Raspbian, PyPI (Python Package Index), and npm repositories. The microcontroller firmware also relies solely on free software libraries. [4]

The HWC circuit board layout and acrylic case design are being prepared for open sourcing as well. When this is done, all aspects of Hedgehog development can be done on the controller itself.

4 Use Cases

As seen in the previous section, Hedgehog is very open about almost all parts of its hardware and software. Some parts are more suitable for advanced users, but many others are accessible to beginners as well.

As Hedgehog is only the controller, we emphasize topics here that focus on that device. For example, combined with an omnidrive chassis, Hedgehog can be used to introduce that mode of motion, along with supporting trigonometric concepts. However, that is mostly due to the chassis, not Hedgehog. We will of course assume that basic robotics components are available.

The following subsections highlight some ways to use Hedgehog in the classroom and other educational settings, ordered by approximate age appropriateness.

4.1 Visual Programming using Pocket Bot and Blockly

As mentioned above, Hedgehog features a visual programming environment in its IDE, and also in the Pocket Bot app. Graphical environments can be used to teach the basics of imperative programming at an early age, and the robot provides a graspable target platform for commands.

Pocket Bot [4] is an extension of the Pocket Code [7] Android app. A Pocket Code program has a stage that is composed of background and objects. Each of these can have scripts that trigger on different conditions, such as the start of the program, or tapping an object.

After the trigger block, command blocks specify actions to execute, such as moving an object on screen. There is a distinction between blocks, which execute
actions and have no result, and functions, which have results and generally no
side effects. Pocket Bot adds blocks for controlling Hedgehog’s motors and servos,
and functions for reading sensor values. Figure 3 shows examples of both. [4]

Figure 2 in the previous section shows the Blockly editor of the Hedgehog
IDE. Control structures are shown in a way that supports the notion that they
group commands, and puzzle-like knobs indicate how blocks can be combined:
commands flow vertically, while expressions are combined horizontally.

The “Code”-button switches the view to a read-only Python version of the
program. Once students are comfortable with formulating algorithms in the vi-
sual environment, they can use this feature to understand how their code trans-
lates to Python, or they can use the visual environment in self-study to remind
themselves on how to write specific pieces of code.

Both visual coding environments provide a slightly limited feature set com-
pared to the Python API: they only support connecting to one controller, and
process management or multithreading features are missing. The assumption
here is that visual coding is mainly used for beginners, where such advanced
features are not needed.

4.2 Textual programming in Python

Textual programming is less appropriate for young children, but has other ben-
efits. We focus here on Python, but these benefits apply to most other program-
ing languages as well.

Python is a multi-paradigm, general-purpose language. It has a large stan-
dard library and ecosystem of third-party packages for various kinds of problems
[4]. It is further regarded as easy to learn [8], with little syntactic overhead.

Fig. 3. Pocket Bot visual programming environment for Android. From left to right:
background and objects in the Pocket Code demo application; Hedgehog command
blocks; part of an object’s script that toggles a motor; and usage of a Pocket Bot
function for sensor readout. [4]
Listing 2 shows a very short Hedgehog program. Compared to a program in C or a related language, there is no main function and all code is executed from top to bottom. Although explaining the exact workings of the first four lines would be too much for a beginner, the `connect()` call at least motivates that the code up to that point sets up a connection for the rest of the program.

```python
from time import sleep
from hedgehog.client import connect

with connect() as hedgehog:
    hedgehog.move(0, 1000)
    sleep(1)
```

Listing 2: Smallest viable Hedgehog program

Although it makes Hedgehog programs look busier at first, we decided against executing setup code before running the user’s actual script. For experienced users, this means that there are no surprises in running one’s code. When a beginner grows out of the core Hedgehog environment, they don’t have to first learn how to imitate the setup that was “magically” done for them before.

### 4.3 Distributed and concurrent applications

The Python Hedgehog library is designed to be threadsafe, and all of Hedgehog is ready to be used in a distributed setup. A simple example of a distributed application would be a remote control, which children love, but is often hard to implement because of the necessary communication.

Listing 1 already showed that accessing two controllers at the same time is very easy. Creating a remote control becomes a matter of reading the sensors of one controller, and effecting the actuators of a second one.

Having different programs to communicate with each other requires more work on the user’s part, but many necessary features are already in place: the discovery system is not limited to advertising the Hedgehog Server, it can be used to find any service on the network. A server also needs to listen for clients’ messages, usually on a new thread; the Hedgehog library’s thread safety is helpful in that regard. Users still need to take care of the usual pitfalls of concurrency, but they don’t need to search library code while looking for problems.

### 4.4 Microcontroller programming

As explained earlier, the SWC comes with a toolchain for the HWC’s STM32F4 preinstalled. In fact, the main way of updating the HWC firmware is to pull the newest source code from GitHub, building it on the Raspberry Pi, and then flashing the HWC.
Compiling code for a microcontroller requires cross-compilation [4], which can be tricky to set up. Having a device that combines development environment and executing device is therefore very convenient, especially when students might bring their own devices with different operating systems, etc.

Users are of course bound to the hardware layout of the HWC [4], but there still is a lot of freedom. The microcontroller’s UART, reset and boot pins are wired to the SWC, and others are connected to the motor drivers or to the servos’ power supply. The microcontroller’s SPI and analog/digital IO pins, for example, can be used freely as with a discovery board.

Users can modify the firmware to do new things, or ignore the HWC and SWC software entirely and create their own microcontroller projects. To restore the original state of the controller, re-flashing the official firmware is sufficient.

4.5 Project participation by students

For PRIA as a joint research and educational organization, Hedgehog is also a valuable tool to include students in our projects. For example, the Hedgehog IDE described in this paper is being developed by a student team from TGM, a technical high school in Vienna, Austria, as part of their graduation project.

Other projects in which high school student teams used or improved Hedgehog technology include “00SIRIS”, where students designed a 3D-printed robot arm for classroom use (shown in figure 4), and “AndriX”, which became the predecessor [9] of Hedgehog.

![Fig. 4. 00SIRIS 3D-printed robotic arm](image)
5 Workshop Experiences

As part of the ER4STEM project, first Robotics workshops using the Hedgehog controller were held. Students were taught the same topics as in previous workshops with controllers from the Botball program, except for the programming language: C in case of Botball controllers, Python in case of Hedgehog.

As of this writing, four Hedgehog workshops with a total of 70 students aged 11 to 16 have taken place. Figure 5 shows metrics for these workshops. For comparison, four Botball workshops with a total of 72 students in the same age range are presented as well. The chart shows response counts relative to group size as the area of the bubbles.

For the difficulty question, there is no significant difference between the kinds of workshops. For the other questions, it can be seen that grades of 5 and 4, or 1 and 2 star ratings, are almost absent from Hedgehog workshops. Of those that gave a grade of 1 or 2, students in Hedgehog workshops more often gave a 1 than those in Botball workshops. For the star rating, this is less clear: two of the four workshops had only 4 and 5 star ratings and a very high proportion of 5 star ratings, while the other two workshops had mostly 4 star ratings.

The absence of bad grades and ratings shows that less students did not engage with the workshops, and the gravitation towards top grades indicates that enthusiasm in those already engaged was greater as well. It has to be noted, how-
ever, that other factors were not corrected for in the data. For example, different people held these workshops, and students from different schools participated in each one.

Oral responses from tutors indicated an overall satisfaction with Hedgehog. Compared to Botball controllers, they say that the system runs more stably, the wireless connection is more convenient than USB, and that the use of Python accelerates the workshops because there is no compilation step.

6 Conclusion and Future Work

First results suggest that Hedgehog in its current state is already a viable teaching tool. A further essential step is the completion of the Hedgehog IDE.

Some teams have registered to use Hedgehog in competitions at the European Conference on Educational Robotics (ECER) 2017 in Sofia, and Hedgehog will be used in workshops throughout the school year and during the summer holidays. All this will give us further feedback and experience reports to improve the system.

Plans for improvements and new features include a change to the HWC circuit board layout to make the controller smaller, APIs for computer vision and controlling iRobot Create robots available at the PRIA lab, and a dataflow programming environment to allow for an alternative to imperative programming.

Acknowledgements. The authors acknowledge the financial support from the European Union’s Horizon 2020 research and innovation program under grant agreement no. 665972. Furthermore, the authors want to thank the high school students who contributed to Hedgehog, in particular Markus Klein, Thomas Fellner, and Hassan Mohamad for their work on the Hedgehog IDE. This is a pre-copyedited version of a contribution published in “Robotics in Education: Latest Results and Developments” edited by W. Lepuschitz, M. Merdan, G. Koppensteiner, R. Balogh and D. Obdržálek in the Advances in Intelligent Systems and Computing book series published by Springer International Publishing AG. The definitive authenticated version is available online via https://doi.org/10.1007/978-3-319-62875-2_21

References

1. LEGO Mindstorms EV3. https://www.lego.com/enus/mindstorms/aboutev3
2. KISS Hardware/Software. http://www.kipr.org/hardwaresoftware
3. Koppensteiner, G., Merdan, M., Miller, D. P.: Teaching botball and researching disbotics. Robotics in Education (2011).
4. Koza, C., Lepuschitz, W., Wolff, M., Frank, D., Koppensteiner, G.: Hedgehog Light – A Versatile, White Box Educational Robotics Controller. Edurobotics 2016: Educational Robotics in the Makers Era, pp. 229-232 (2016). doi:10.1007/9783319555539_19
5. Hintjens, P.: ZeroMQ: Messaging for Many Applications. O’Reilly Media, Inc., Sebastopol (2013).
6. Protocol Buffers. https://developers.google.com/protocolbuffers/docs/overview
7. Catrobat Pocket Code. https://www.catrobat.org
8. Donaldson, T.: Python as a first programming language for everyone. Western Canadian Conference on Computing Education (2003).
9. Koza, C., Krofitsch, C., Lepuschitz, W., Koppensteiner, G.: Hedgehog: A Smart Phone Driven Controller for Educational Robotics. 6th International Conference on Robotics in Education, pp. 50-52 (2015).