On Computational Notebooks to Empower Physical Computing Novices

On Computational Notebooks to Empower Physical Computing Novices

FULVIO CORNO, Politecnico di Torino, Italy
LUIGI DE RUSSIS, Politecnico di Torino, Italy
JUAN PABLO SÁENZ, Politecnico di Torino, Italy

The ever-increasing availability and variety of resources to create physical computing systems keep attracting electronics hobbyists and do-it-yourself enthusiasts. Nevertheless, the prototyping and development of these systems are still challenging to the novices. In this paper, we propose a tool (built on top of the Jupyter computational notebook) as a way for supporting step-by-step assisted learning and knowledge sharing. We extended the Jupyter notebook functionalities and implemented a custom-tailored kernel to seamlessly enable the interaction between the end-user web interface and the Arduino boards. We consider that this approach can effectively support physical computing novices in understanding, writing, and executing the code while empowering them to document and share the development steps they followed.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; • **Software and its engineering** → *Development frameworks and environments.*

Additional Key Words and Phrases: Computational Notebook; Kernelino; Physical Computing; Arduino

ACM Reference Format:
Fulvio Corno, Luigi De Russis, and Juan Pablo Sáenz. . On Computational Notebooks to Empower Physical Computing Novices. In . ACM, New York, NY, USA, 6 pages.

1 INTRODUCTION

Arduino is a well-known, widely-adopted physical computing platform. It is very popular among do-it-yourself (DIY) enthusiasts and has encouraged end-user developers from diverse disciplines to create physical computing artifacts. However, getting started in the development of this kind of projects remains challenging for novices: they must learn and apply both programming and electronics concepts while developing an understanding of the relationship between the software and hardware of their systems [2].

The process of going from schematics and code posted on the web to a functional physical computing artifact is fraught with pitfalls, and step-by-step tutorials can lead users to perform actions without fully understanding their intention [14]. Consequently, custom coding tools and knowledge sharing mechanisms, able to provide in-context guidance during the construction process while educating in good practice, have been raised as an essential means to empower amateurs and hobbyists to become designers and producers [7, 10].

Furthermore, these coding tools and knowledge sharing mechanisms need to consider novices *just-in-time* designs and assist them in making and documenting their modifications iteratively [3]. In this scenario, we observed that in the field of data science, computational notebooks are a widely-used tool that enables data scientists to accurately document, execute, and share the steps of their analyses in an exploratory and iterative manner. They support the

Manuscript submitted to ACM

consolidation and sharing of computational narratives by consolidating text, executable code, and visualizations in a single document [9].

The contribution of this work is in providing a tool that extends the current computational notebooks' capabilities to support novice programmers in becoming proficient in coding and executing physical computing systems while empowering them to document and share the steps they followed. To that end, we developed two artifacts on top of the JupyterLab computational notebook: (i) an extension with new features and custom end-user interface components; and (ii) a custom-tailored kernel (Kernelino) that allowed a seamless interaction between an Arduino board and a computational notebook.

2 RELATED WORKS

Numerous tools have been proposed to enable rapid prototyping of physical computing systems. The following are some software-based tools aimed at supporting novice developers.

Inspired by test-driven development practices in Software Engineering, in [14] the authors present a test-driven physical computing tutorials. These tutorials include interactive tests on each step, requiring the user to verify wiring, electronics, code, or their own knowledge before they can proceed to subsequent steps. According to the authors, an important area for future work is to examine how to support authors in creating test-driven tutorials. They suggest that there may be advantages in enabling tutorial users, or their peers, to create tests as well, with the intent of promoting a deeper reflection and understanding of the tutorial content.

By adopting a Generative Design approach, Anderson et al. [1] introduce TAC (Trigger-Action-Circuits), a system supporting novice users in the design and assembly of functional electronic devices. It generates circuitry, firmware, and assembly instructions from a user's high-level description of the desired functionality. Following assembly, the user may inspect the code to gain an understanding of the underlying program. As the authors acknowledge, while this tool can generate circuits and software, neither the code nor the generated diagram is suitable for learnability. Additionally, as the circuit and software generation is an automated process, it isolates the user from how the circuit works.

In [4], the authors present a general-purpose conversational programming system to support makers in building a physical computing prototype. It guides users during hardware assembly by providing additional information on requests or by interactively presenting the assembly steps to build a circuit. The user can program and execute code in real-time on their Arduino platform without having to write any code, but instead by using commands triggered by voice or text via chat. Authors state that vocal commands reduce the cognitive load needed to focus on two different tasks: programming and assembling hardware.

Our proposal differs from the previously described works in two main respects. On the one hand, since we aim to help novices become proficient in coding and implementing physical computing systems, our tool is not intended to isolate novices from the code, provide higher-levels of abstraction, or automate code-generation. On the contrary, we are motivated by the idea that becoming familiar with existing code and understanding its implicit problem-solving strategy (thanks to a rich surrounding context) leads to learning better ways to code, access superior knowledge, and expand the culture and practices of DIY [12, 13]. On the other hand, besides providing in-context guidance to the novices through the implementation process, our proposal focuses on empowering them to document and share the steps they followed when developing their physical computing systems. Therefore, by relying on a widely-adopted end-user-oriented platform, our goal is to overcome the reported lack of tools to document the novice's work [5], and as currently occurs in the field of Data Science, allow them to record, share, and reproduce the development process easily.

3 COMPUTATIONAL NOTEBOOKS

Computational notebooks have emerged as a mean to support the construction and sharing of computational narratives by enabling data analysts to arrange code, visualizations, and text in a computational narrative [9, 11]. They are based on cells containing rich text or executable code that generates results or visualizations. These cells are linearly arranged but can be reorganized, reshuffled, and executed multiple times in any order. Among the currently available computational notebooks, Project Jupyter is one of the most widely used platforms [6], it is a popular open-source computational notebook that relies on open standards and enables users to combine code, visualizations, and text in a single document whose underlying structure is JSON.

Technically speaking, the term *computational notebooks* commonly refer at the same time to the interactive literate programming documents and to the software platform to execute them [8]. The *computational notebooks* platform consists of: a kernel that runs the code cells in a particular programming language and returns the corresponding output to the user; and an interactive computing protocol that standardizes and manages the communication between the *notebook documents* and the kernels. A *kernel* for its part is a "computational engine" that executes the code contained in the code cells of a Notebook document. When the notebook document is executed, the kernel performs the computation and produces the results.

Additionally, a *notebook document* consists of a JSON document containing text, source code, rich media output, and metadata. At the highest level, a notebook is a dictionary with a few keys: metadata (dict), nbformat (int), nbformat_minor (int), and cells (list). There are two types of cell types; markdown cells and code cells. The former ones contain source code in the document's associated kernel language and a list of outputs associated with executing that code. In short, a *notebook document* consists of a file with descriptive text cells interleaved with executable code cells.

Finally, *JupyterLab* is a next-generation web-based user interface for Project Jupyter. It is extensible and modular. Hence, it allows developers to write plugins that add new components and integrate with existing ones. In this manner, as described below, an extension comprises one or more plugins that extend JupyterLab.

4 JUPYTERLAB + ARDUINO

To achieve the integration between a computational notebook and an Arduino board, we developed a tool composed of two software artifacts. A JupyterLab plugin adds new features and customizes the end-user interface, and a custom tailored-kernel (the *Kernelino*) enables the integration between the board and the computational notebook. We discuss each of these artifacts in turn. Furthermore, Figure 1 provides an overview of the implemented architecture components.

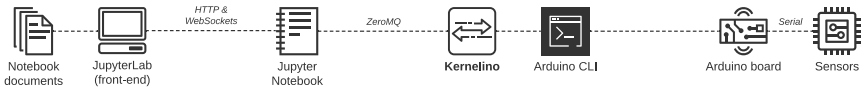

Fig. 1. Overview of the implemented architecture components

4.1 JupyterLab Extension

The JupyterLab application is comprised of a core application object and a set of extensions. These extensions provide nearly every function in JupyterLab, including document editors and viewers, code consoles, terminals, themes, the file browser, and settings editor. Extensions even provide more fundamental parts of the application, such as the menu system, status bar, and the server's underlying communication mechanism. Through the extensions, developers can

customize or enhance any part of JupyterLab, including new themes, file editors, and custom components. In sum, JupyterLab extensions add features to the user experience. Furthermore, an extension contains one or more plugins that extend JupyterLab.

In our tool, we took advantage of the JupyterLab modular design and implemented an extension to add the following features:

- **Launch a custom Arduino notebook document**: as shown in Figure 2, once Jupyter is deployed, the JupyterLab web interface now gives the users the option to launch an Arduino notebook document. Such Arduino notebook document is linked by default with the *Kernelino* (described later in this section), who manages all the code cells execution requests and retrieves their corresponding output.

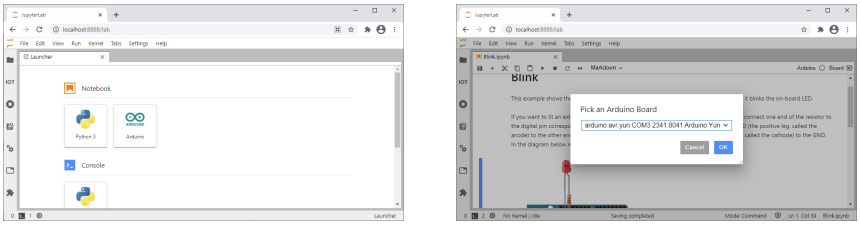

Fig. 2. Launcher to create an Arduino document (left) and modular menu to choose the Arduino board (right)

- **Select on which board to execute a notebook document**: as illustrated in Figure 2, once created a new Arduino notebook document, the tool lets the user choose on which board to execute the code cells. This selection is mandatory, and all the run buttons in the code cells remain disabled until the user selects the Arduino board and the custom-tailored kernel achieves the connection. As also shown in Figure 2, the button to choose the board is in the right upper corner of the document. When successfully connected, the label changes from "Board" to the board's model name ("Arduino Yun" in this case).
- **Extend the default notebook's code cells to map the Arduino's sketch structure**: Users can indicate on each code cell whether the code on it belongs to the `setup` or `loop` methods or a custom method. We implemented this feature by extending the default cell footer, adding radio buttons where the users can specify the respective method, as shown in Figure 3.
- **Generate and export all the code cells as an Arduino file (`.ino`)**: When the user executes a code cell for the first time, the Kernelino creates and saves an Arduino sketch file. After this, as the user runs a code cell, the tool modifies the sketch adding the corresponding code in the indicated method. In this manner, the development of physical computing becomes incremental. As the user advances through the documentation and executes the cells, the sketch grows. This strategy enables to document and run the code very accurately. A user might write an explanation or add images or figures for every single line of code, as shown in Figure 3. Additionally, users are free to save and share either the notebook document from which an Arduino sketch can be wholly reconstructed or just the sketch file.
- **Modify the underlying notebook's JSON file**: In line with the newly added functionalities, we modified the notebook's underlying JSON schema to model the method to which each code cell belongs. When the user saves the notebook document, a new field is added into the JSON dictionary representing a code cell. Naturally, this modification persists for the next time the user works on the document.

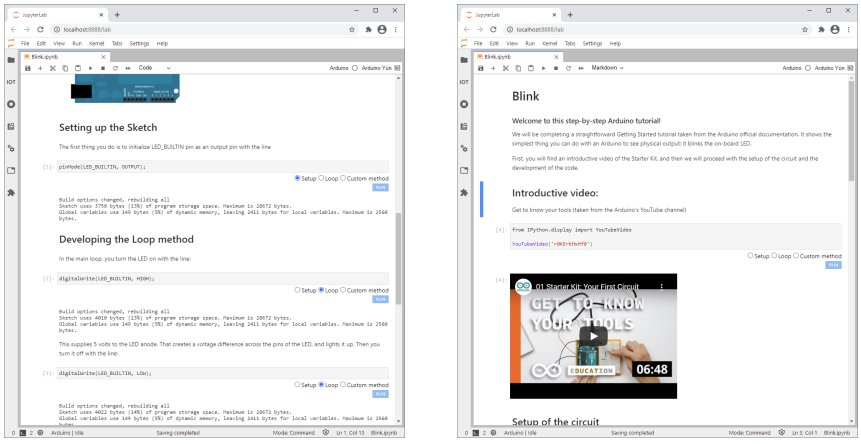

Fig. 3. Custom cells to map the Arduino's sketch structure (left) and surrounding contextual information in the physical computing development process (right)

It is essential to highlight that these new functionalities integrate with the standard JupyterLab features. Consequently, as shown in Figure 3, the physical computing development gets incremental and is enriched with various types of surrounding contextual information (as might be diagrams, links, photos, links, and videos). Additionally, the development process can be easily captured and shared through a format that combines executable code and documentation in a platform that supports the reconstruction of the whole implementation steps.

4.2 Jupyter Kernelino

As illustrated in Figure 1, the *Kernelino* is a custom-tailored kernel that behaves as an intermediary between the Jupyter application and the Arduino board. It is accountable for: (i) Gathering the list of boards physically connected to the computer on which the computational notebook is running. (ii) If it does not exist, generating an Arduino sketch file every time the user executes a code cell. (iii) Injecting the code in the sketch's right position, according to the cell's method (setup, loop, or custom method). (iv) Compiling and executing the resulting sketch by invoking the Arduino command-line interface (CLI) corresponding command. (v) Retrieving the output from the Arduino CLI to the Jupyter application.

The implementation of the *Kernelino* required to integrate the messaging protocols of Jupyter with the Arduino CLI. However, since the messaging protocols are complex, writing a new kernel from scratch is not straightforward. For this reason, we used an interface provided by Jupyter to wrap kernel languages in Python. We subclassed `ipykernel.kernelbase.Kernel` and implemented the methods and attributes that forward the code from the Jupyter notebook to the Arduino CLI and retrieve the corresponding response. In short, the interface provided by Jupyter handles all the ZeroMQ (a high-performance asynchronous messaging library) sockets and communication mechanisms, making sure that the messages are correctly created and parsed for each type of request between the Jupyter notebook component and the Kernelino.

5 CONCLUSION AND FUTURE WORK

This article presented a tool to support novice programmers in the field of physical computing development. Specifically, we extended and added new capabilities to the current computational notebooks. By constructing and sharing computational narratives, developers who are approaching the first time physical computing development can fully understand the design and development process and, most importantly, become proficient with the coding and share their implementations with other novices. Future research concerns the evaluation of the tool with inexperienced students working in real-world physical computing projects. We will evaluate the tool's usability and assess to what extent it effectively supports the novices helping them overcome the issues related to understanding the problem.

REFERENCES

[1] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 331–342. https://doi.org/10.1145/3126594.3126637

[2] Tracey Booth, Simone Stumpf, Jon Bird, and Sara Jones. 2016. Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '16)*. Association for Computing Machinery, New York, NY, USA, 3485–3497. https://doi.org/10.1145/2858036.2858533

[3] Kayla DesPortes and Betsy DiSalvo. 2019. Trials and Tribulations of Novices Working with the Arduino. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 219–227. https://doi.org/10.1145/3291279.3339427

[4] Yoonji Kim, Youngkyung Choi, Daye Kang, Minkyeong Lee, Tek-Jin Nam, and Andrea Bianchi. 2019. HeyTeddy: Conversational Test-Driven Development for Physical Computing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 4, Article 139 (Dec. 2019), 21 pages. https://doi.org/10.1145/3369838

[5] Stacey Kuznetsov and Eric Paulos. 2010. Rise of the Expert Amateur: DIY Projects, Communities, and Cultures. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* (Reykjavik, Iceland) *(NordiCHI '10)*. Association for Computing Machinery, New York, NY, USA, 295–304. https://doi.org/10.1145/1868914.1868950

[6] Sau Lam, Ian Drosos, Julia M. Markel, and Philip J. Guo. 2020. The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–11. https://doi.org/10.1109/VL/HCC50065.2020.9127201

[7] David A. Mellis, Leah Buechley, Mitchel Resnick, and Björn Hartmann. 2016. Engaging Amateurs in the Design, Fabrication, and Assembly of Electronic Devices. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems* (Brisbane, QLD, Australia) *(DIS '16)*. Association for Computing Machinery, New York, NY, USA, 1270–1281. https://doi.org/10.1145/2901790.2901833

[8] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A Large-scale Study About Quality and Reproducibility of Jupyter Notebooks. In *Proceedings of the 16th International Conference on Mining Software Repositories* (Montreal, Quebec, Canada) *(MSR '19)*. IEEE Press, Piscataway, NJ, USA, 507–517. https://doi.org/10.1109/MSR.2019.00077

[9] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. ACM, New York, NY, USA, Article 32, 12 pages. https://doi.org/10.1145/3173574.3173606

[10] Theresa Jean Tanenbaum, Amanda M. Williams, Audrey Desjardins, and Karen Tanenbaum. 2013. Democratizing Technology: Pleasure, Utility and Expressiveness in DIY and Maker Practice. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Paris, France) *(CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2603–2612. https://doi.org/10.1145/2470654.2481360

[11] Mauricio Verano Merino, Jurgen Vinju, and Tijs van der Storm. 2020. Bacatá: Notebooks for DSLs, Almost for Free. *The Art, Science, and Engineering of Programming* 4, 3 (Feb 2020). https://doi.org/10.22152/programming-journal.org/2020/4/11

[12] Camilo Vieira, Alejandra J. Magana, Michael L. Falk, and R. Edwin Garcia. 2017. Writing In-Code Comments to Self-Explain in Computational Science and Engineering Education. *ACM Trans. Comput. Educ.* 17, 4, Article 17 (Aug. 2017), 21 pages. https://doi.org/10.1145/3058751

[13] Ron Wakkary, Markus Lorenz Schilling, Matthew A. Dalton, Sabrina Hauser, Audrey Desjardins, Xiao Zhang, and Henry W.J. Lin. 2015. Tutorial Authorship and Hybrid Designers: The Joy (and Frustration) of DIY Tutorials. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 609–618. https://doi.org/10.1145/2702123.2702550

[14] Jeremy Warner, Ben Lafreniere, George Fitzmaurice, and Tovi Grossman. 2018. ElectroTutor: Test-Driven Physical Computing Tutorials. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 435–446. https://doi.org/10.1145/3242587.3242591