IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 14, 2020, accepted October 26, 2020, date of publication November 2, 2020, date of current version November 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3035083

BIPES: Block Based Integrated Platform for Embedded Systems

ANDOUGLAS GONÇALVES DA SILVA JUNIOR[1], LUIZ MARCOS GARCIA GONÇALVES[2], (Member, IEEE), GLAUCO A. DE PAULA CAURIN[3], (Member, IEEE), GUSTAVO TERUO BERNARDINO TAMANAKA[3], ANDRÉ CARMONA HERNANDES[4,5], AND RAFAEL VIDAL AROCA[4,5], (Senior Member, IEEE)

[1]Academic Directory, Instituto Federal do Rio Grande do Norte (IFRN), Mossoró 59628-330, Brazil
[2]Graduate Program on Electrical and Computer Engineering, Universidade Federal do Rio Grande do Norte (UFRN), Natal 59078-970, Brazil
[3]Aeronautical Engineering Department, School of Engineering of Sao Carlos (EESC), Universidade de São Paulo (USP), São Carlos 13566-590, Brazil
[4]Computer Science Department, Universidade Federal de São Carlos (UFSCar), São Carlos 13565-905, Brazil
[5]Electrical Engineering Department, Universidade Federal de São Carlos (UFSCar), São Carlos 13565-905, Brazil

Corresponding author: Luiz Marcos Garcia Gonçalves (lmarcos@dca.ufrn.br)

This work was supported in part by the National Council for Scientific and Technological Development (CNPQ) under Grant 309447/2017-8, Grant 314936/2018-1, and Grant 141395/2017-6; in part by the São Paulo Research Foundation (FAPESP) under Grant 2017/01555-7; and in part by the Coordination for the Improvement of Higher Education Personnel (CAPES) under Grant 001.

ABSTRACT This article proposes the BIPES, a **B**lock based **I**ntegrated **P**latform for **E**mbedded **S**ystems, including its architecture, design and validation results. BIPES is an open source software and service that is freely available through the website http://www.bipes.net.br and has been conceived from our experience of several years developing embedded systems and Internet of Things (IoT) applications, and teaching. It allows anyone to quickly and reliably design, program, build, deploy and monitor embedded systems, IoT devices and applications using blocks or Python based programming. It is fully based on web environment, so absolutely no software installation is needed on the client developer machine. In this way, a tablet, a netbook, a Chromebook or any other device can be used to program and test several types of devices. Mainly, it relies on MicroPython or CircuitPython, WebREPL, WebSockets, Web Serial API, HTML, JavaScript and Google Blockly to allow no-code programming (blocks) to be translated into Python code and then deployed to the target board. Moreover, it does not require server side processing, so it can be deployed as a Progressive Web Application (PWA), allowing it to be used even when the computer is offline. It is compatible with several low cost boards such as: mBed, BBC micro:bit, ESP8266, ESP32 and Raspberry Pi using only a web browser and without the need to install any software on the device where the user develops the programming.

INDEX TERMS Embedded systems, block based programming, Internet of Things, STEAM, wearable devices programming.

I. INTRODUCTION

Programming Without Code: The Rise of No-Code Software Development, a recent article by Rina Diane Callabar [6] on IEEE Spectrum Tech Talk section remembers how important software is and how several tools are helping more people to program without code. In that way, several tools and approaches are already available for people interested in developing several types of software without the need of writing code. One of the most common alternatives to the traditional code based programming is block based programming, which is intuitive for several users.

A successful example of block based programming is the MIT App Inventor (https://appinventor.mit.edu/), a consolidated and freely available online tool that allows easy development of smartphone applications using drag and drop and block based programming. According to MIT App Inventor usage information from the project website (July/2020), App Inventor already has more than 8 million users, from 195 countries and more than 30 million smartphone applications were developed using App Inventor.

In fact, we have been teaching programming concepts using App Inventor for Brazilian students of several education

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaolong Li.

FIGURE 1. Example of soccer game application developed using MIT App Inventor by a 13-year old student.

levels and observed that even students that had never programmed before can create and customize their own applications in about 4 hours. Figure 1 shows an example of simple and fully functional soccer game created by a 13-year old student, during a 4-hour workshop - we notice that this student had never programmed before.

The block based programming approach used by App Inventor is key for teaching programming, Science, Technology, Engineering, Arts and Math (STEAM), and indeed it foster students interest, motivation and empowerment, being usually chosen as entry-point language [2] and usually have a better performance with non-experts in programming [5]. More than that, it has been enabling enthusiasts to build real world prototypes, applications, innovations and business, such as XLBlocks [3], for spreadsheet formulas editor and CoBlox [4], a block-based programming interface for a single-armed industrial robot.

A special type of software, often called embedded software, is present in hundreds of types of products, such as printers, cameras, televisions, robots and cars. In fact, about 1/3 of the costs of cars and airplanes are related to the product's electronics, embedded systems [7], and consequently embedded software operating in such products.

A related definition is "Physical Computing", which combines software and hardware "to build interactive physical systems that sense and respond to the real world" [1]. According to Hodges *et al.* [1] there are still challenges and work to be done in such area and a close integration between hardware and software is critical to a good user experience. Such challenges motivated our applied research, which resulted in the development of BIPES.

Moreover, with the recent increasing interest in Robotics, IoT, and embedded systems in all levels of education, there is still a lack of simple and intuitive tools to program such devices either for education or even professional usage. In this way, most embedded programming tools, even the ones that allow block based programming show some restrictions. Most of them require software download and installation, configuration and boards with specific firmware, which are hard to find or expensive. Moreover, some of these tools are commercial and a licence must be purchased in order to be used.

Yet in this direction, we have conducted 4-hour workshops with students, using NodeMCU (ESP8266) modules for IoT, and we observed some common characteristics in most sessions. One issue observed is the requirement of installation of compilers and integrated development platforms (IDEs), and configuring the correct board and device drivers that usually took about 25% of the workshop duration. Moreover, we also observed platform differences and incompatibilities (Linux, Mac, Windows), or even permissions issues. These difficulties are also reported by the authors of Microsoft MakeCode platform, mentioning that schools have a diversity of computers and that "the only software in common across platforms is the modern web browser" [24]. Moreover, in corporate computers or school laboratory computers, software installation and configuration requires administrative permission, which cannot be obtained in many cases.

Such situations motivated us to propose a new approach that we named BIPES, which, by design, consists of a fully web-based tool that does not require any software installation, nor even add-ons or plugins for the web browser. Another project goal is to allow block based programming with multiple target boards and modules. Yet, it also works offline as a web application, which is important for a good user experience as some trials of BBC micro:bit in UK schools using cloud compilation had "unacceptable failures due to spotty network connectivity" [24]. In that way, BIPES does not compile code, but generates Python code to be interpreted by a variety of boards running MicroPython, CircuitPython or standard Python. So, the only requirement for BIPES to work is a device with a web browser and a board with MicroPython or CircuitPython previously loaded (only one time), or an embedded Linux device with Python interpreter installed. Such boards can be purchased with such firmwares pre-loaded, or can be flashed by the user or someone that prepares new boards before workshops.

Therefore, the main contributions of this paper are to introduce the design, architecture, implementations, and current status of BIPES development, and making it fully available to other groups and researchers as an open source tool. It relies on MicroPython, CircuitPython or Python running on target boards, such as ESP32, ESP8266, Raspberry Pi or even a standard x86 computers. In order to interact with the boards, it uses W3C Web USB API communication to talk

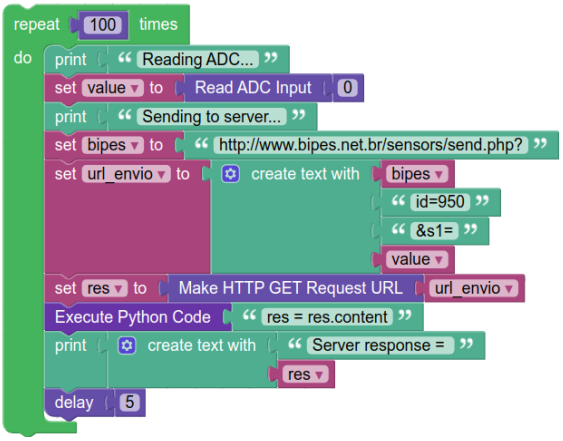

FIGURE 2. Complete IoT example developed with BIPES.

directly from the browser to the devices using USB cable or WebREPL and WebSockets to communicate with the devices using network. It also integrates Google Blockly and other open-source softwares. In such a way, it is possible to design programs using blocks, and quickly transfer, run and monitor it from modern web browsers. Figure 2 shows an example of a program built in a few minutes, which can read an analog sensor and send the obtained data, in real time to a server, which allows data visualization, in a real time graph and keeps the sent data in a database for history analysis. This test is conducted with an ESP8266 board. In an extensive search we have not found any similar architecture, and with such features. So we truly believe that this new architecture model and the BIPES itself are a very good contribution not only to educational purposes but also to more general automation research development.

The article is organized as follows: section 2 discusses related works and the state of the art. Section 3 introduces BIPES underlying technologies, which provides the infrastructure for BIPES to work. Section 4 proposes the BIPES architecture and design. Section 5 discusses the BIPES current development stage and some study cases, and section 6 presents our final remarks including ongoing and future works.

II. RELATED WORKS

Code implementation is considered one of the most important step in software development life-cycle [25]. However, from software design to real implementation, a semantic gap [23] may exist due to several factors, such as communication and developers interpretation of the problem to be solved. Moreover, text based programming is prone to possible mistakes, which are not acceptable for some applications: for example, in safety-critical systems, ''failure is not an option'' [26]. On the other hand, studies show that code based software may have up to 50 errors for each 1000 lines of code [27].

In this direction, engineering applications have been using approaches with automatic code generation, such as model driven engineering (MDE) and model driven development (MDD), with automatic code generation (ACG), which has been proven to reduce source code writing and leading to considerable reduction in development time [28]. Actually, Liao *et al.* [22] argue that ACG improves software quality and shortens development cycles.

With the increasing demand for developers and new applications, automatic code generation has been gaining each time more importance, and now such tools are mostly known as no-code or low-code programming tools [6]. However, such tools are typically used by developers and engineers. With the scenario of low cost devices, several efforts are being done to take no-code and low-code programming to a wider public. Several tools are already available, such as Snap!, MicroBlocks, Scratch for Arduino, Blocklyduino, and Microsoft Make Code, between others.

The above mentioned App Inventor has also generated spin-off projects related to embedded and IoT devices programming. In fact, a block-based programming approach using MIT App Inventor [15] was proposed in order to be able to help novice programmers to build mobile apps in a simpler way, integrated with IoT technology. Basically, the authors used the MIT platform to create Arduino algorithms blocks, such that the user can just drag and use these blocks without necessary understand the math behind them. Such project is focused on Arduino 101. App Inventor IoT Embedded Companion is another project [19] that adds more functionalities to the App Inventor in order to make easier the development of Arduino projects using light sensor, humidity and temperature sensor, buzzer, and others. For such project, also, an Arduino with a special code interpreter is needed.

The soEasy project by Lee *et al.* [8] uses a web-based IDE and a database to configure pins and alternate functions on processors that was included on the same database. The ease of configure pin has a similar idea that CubeMx for ST microcontrollers. A downside is that the tool uses C/C++ code implementation despite being block-based. Hauck and colleagues [12] have chosen ArduBlock as block-based programming IDE and used a MQTT communication to a IBM node-RED device to run AI applications, thus, the device that captures the signal does not process it. Other proposals including MicroPython and Blockly are also found for embedded programming [13], [14]. In those study cases, they use Chrome Embedded Framework to build a desktop program based on a web application and then uses serial communication to load the program.

Analyzing the challenges brought by recent advances on Industrial Internet of Things [10], one of them is attending multi-vendor technologies for Industrial Internet-of-Things systems. As observed until now, neither applications have this cross vendor/platform capability, which the proposed BIPES architecture has.

The DSCBlocks team [9] has created a web-based block-based programming IDE focused on dSPic microcontrollers. TUNIOT is as a web based tool (http://www.easycoding.tn/tuniot/demos/code/), that makes possible to program the

ESP8266 module (NodeMCU) in blocks, and the tool generates a program in C language, from blocks prepared using Blockly, but does not send or compile the code. To compile and send the code to the board, it is necessary to install a compiler on the local computer and firmware flashing tools on the user's local computer. ArduBlockly (https://github.com/carlosperate/ardublockly), focused only on the Arduino platform, also allows block based programming, but it is necessary to install local software (server) on the user's PC to compile and send the program to the Arduino.

Several other projects are worth mentioning as MicroBlocks (https://microblocks.fun/tech), which provides a kernel/firmware for quickly programming embedded boards, however software download is needed for programming on the desktop. Snap4Arduino (http://snap4arduino.rocks/) is another block based programming environment for most Arduino boards, but also, it consists of a desktop application, needing download and installation. On the same way, Scratch for Arduino (http://s4a.cat/) also allows block based programming focused on Arduino boards, but requires download and installation of desktop software and a firmware on target Arduino boards.

At this point we discuss in more detail the UIFlow and Microsoft MakeCode projects that are the ones mostly related to BIPES. UIFlow is a web-based programming environment (https://flow.m5stack.com/) that was created focused on programming m5stack / m5stick commercial products based on ESP32 processor running MicroPython. It allows block based programming using Blockly, and blocks are converted into Python code. As it is focused on the m5 product family, it requires an API key to connect to these specific boards.

In the MakeCode and CODAL project [11] the authors use a web-based block-based programming IDE for microcontrollers and use UF2 as bootloader, which makes the board act as USB thumb drives when connected to a computer. In that way, to run a program on the board, a user simply has to copy the compiled file to the board as a pen drive. This approach is also used by mBed environment and online editor/compiler (https://os.mbed.com/ide/).

The Microsoft MakeCode [24] is an open source (MIT licence) web-based block based environment supporting several target platforms, such as Lego Mindstorms EV3 and BBC micro:bit. It requires no software installation and works offline after its been loaded in web browsers. It supports WebUSB, which allows direct browser communication with USB devices, however this is only possible with boards compatible with WebUSB. On the other hand, this it is not a requirement as the UF2 bootloader can be used for most computers / boards.

This review has shown that many research works have been done in the context of block based programming for embedded devices, and several projects are very active, such as UIFlow and Microsoft MakeCode. Nonetheless, there are several further possibilities to be explored, as for example the capability of attending every device without need of any software installation, between others. In this context, BIPES brings some additional contributions to this ecosystem with some differentials that will be detailed further in Section IV.

III. BIPES UNDERLYING TECHNOLOGIES

As said, this paper goal is to introduce the design, architecture, implementations, and current status of BIPES development. To make it fully available to other groups and researchers, it is necessary to explain some related technology as Google Blockly, MicroPython, and CircuitPython (or Python running on target boards). In order to interact with the boards, direct communication from the browser is achieved through the W3C Web USB API using USB cable or WebREPL and WebSockets. Thus, details of these used technologies will be introduced and discussed in this Section to aid the understanding of their use to design and program using blocks, and quickly transfer, run and monitor it straight from web browsers.

A. GOOGLE BLOCKLY

Google Blockly is JavaScript library for building visual programming editors to create Visual Programming Languages (VPL). Created by Google in 2012 and also maintained by Google [16], [17], Blockly has already been used in hundreds of projects and applications. MIT App Inventor, for example, uses Blockly. Blockly provides features to build block based toolboxes, to connect the blocks according to patterns defined by the developers and generate source code automatically for several languages. Blockly also allows blocks to be imported, exported and shared using XML file format. BIPES relies on Blockly to allow users to built their programs using blocks and then automatically generate Python source code.

Although Blockly is widely used in educational contexts, there are also study cases of Blockly applied in commercial / industrial situations. Weintrop *et al.* [18], for example, presents the usage of Blockly applied to industrial robotics programming. They argue that they conducted a small-scale study and observed that Blockly enabled ''adults with no prior programming experience successfully programming a virtual robot to accomplish a pick and place task''.

B. MicroPython OR CircuitPython

Python is an Open Source high-level programming language, with a growing community and user base. It is used in educational contexts and to develop real-world applications in the industry, services and even for scientific computing.

MicroPython (https://micropython.org/) is a complete Python compiler and runtime, optimized for devices with low resources, that runs on the several low cost microcontrollers. It is mostly compatible with Python 3 and provides an interactive prompt (REPL). It was originally written by Damien George, having its first release in 2014. Among several supported devices, we highlight some ARM families, STM32, ESP32 and ESP8266, 16-bit PIC, and a native Unix port (running on a Raspberry / ARM or x86 Linux).

Another relevant tool is CircuitPython: according to its website (https://circuitpython.org/), ''CircuitPython is a

programming language designed to simplify experimenting and learning to code on low-cost microcontroller boards''. Created in 2017, CircuitPython is a fork of MicroPython intended to be a solution for microcontroller programming, which is usually done with C, C++ or Arduino. It includes a selection of core Python libraries and modules which give the programmer access to the low-level hardware of several microcontrollers.

Both MicroPython and CircuitPython allow a direct read–eval–print loop (REPL) interactive shells over serial (USB) port. REPL works in a similar way to a shell, where a user may enter commands to a prompt, which are executed and the result shown to the user. An example of REPL, is accessing an interactive Python session, where the user may define variables, call functions, and interact with the interpreter.

C. WebSocket AND WebREPL

WebSocket is a World Wide Web Consortium (W3C) recommendation / specification, implemented as Application Programming Interfaces (APIs) in modern web browsers. It enables Web pages to use the WebSocket protocol (defined by the IETF) for two-way communication with a remote host, using JavaScript, for example. In that way, a standard HTML page / site, without any server support may start and keep direct full-duplex communication with devices or other servers.

WebREPL consists of a REPL session over WebSockets, in a way that the client is a standard HTML5 application running on the web browser, talking with the board using network (Wifi, for example). WebREPL not only provides an interactive REPL over the network, but has also defined a protocol to send and receive files and other commands to the target board through the network. In that way, WebREPL allows storing and retrieving files from a remote device filesystem, starting programs, and interacting with programs in remote boards directly from a standard HTML application.

D. WEB SERIAL API

It is known that many applications and management systems are each time more being transformed into web-based cloud solutions, fully accessed and managed from any web browser. In fact, many corporate systems are web based. However, one typical past limitation of web browsers is to directly access the hardware, specifically USB devices. In this way, a recent effort from the community led to the W3C Draft Community Group Report (13 August 2020) of Web Serial API, which proposed a preliminary specification for the implementation of web browser direct communication with devices.

According to such document (https://wicg.github.io/serial/), the Serial API provides a way for websites to read and write from a serial device through script. Such an API would bridge the web and the physical world, by allowing documents to communicate with devices such as microcontrollers, 3D printers, and other serial devices. There are also other specifications as WebUSB (https://wicg.github.io/webusb/) or Web Bluetooth (https://webbluetoothcg.github.io/web-bluetooth/) however they need specific boards and chipsets to communicate. Also, the Web Bluetooth API context has more focus on device attack from malicious websites, thus direct access to some Bluetooth features are eventually removed. Actually, it needs the user to select and approve the access to devices. In that way, one advantage of Web Serial API is the possibility to communicate with any serial or USB-Serial device, without the need of specific chipsets or board configuration.

E. PROGRESSIVE WEB APPLICATIONS

As mentioned in the last section, several applications are web-based today. With modern web-based technologies, such as HTML5, JavaScript, storage, WebSockets and others, web technologies became a full featured environment to develop several kinds of applications, that can be fully executed from the browser, using the scripting languages that web browsers can execute, specially JavaScript.

Such applications are typically stored and deployed through web servers. Thus, a natural step further is to download the set of files of an application stored in a web server to execute it directly from local files stored on the user's device. Progressive Web Applications (PWA) [20] allows such operation, but in a easy and practical way. Basically, a PWA are simple websites that can be locally installed, instead of being downloaded from an App Store.

As PWAs are standard HTML5 applications, they are platform independent and can provide interactions similar to real Apps from standard App stores. They can work offline, and are easy to install. When the a user access a PWA compatible website / application, an ''Install'' button is automatically shown, and the user can install it directly from the browser.

BIPEs relies on PWA, allowing the BIPES block programming environment to be eventually installed to be used offline, for example, on laboratories or schools with unstable Internet connection.

F. OTHER TECHNOLOGIES

Other technologies are also involved to enable BIPES functionality, which we mainly highlighted, as Google Cloud Storage to store and share BIPES programs, and the HighCharts API to plot IoT real time and historic graph. An interactive console terminal is also used (term.js), which is written in JavaScript, and the OpenCV on Raspberry Pi library to allow computer vision applications to be developed using BIPES blocks.

IV. BIPES ARCHITECTURE

The general overview of BIPES components is shown in Figure 3. Block based programs can be developed by opening BIPES web application, either online of offline (web application stored on disk). After the program is ready, the user can connect to a device using USB or network to send, monitor and control the created program using WebREPL protocol over network or Serial REPL over Serial-USB interface.

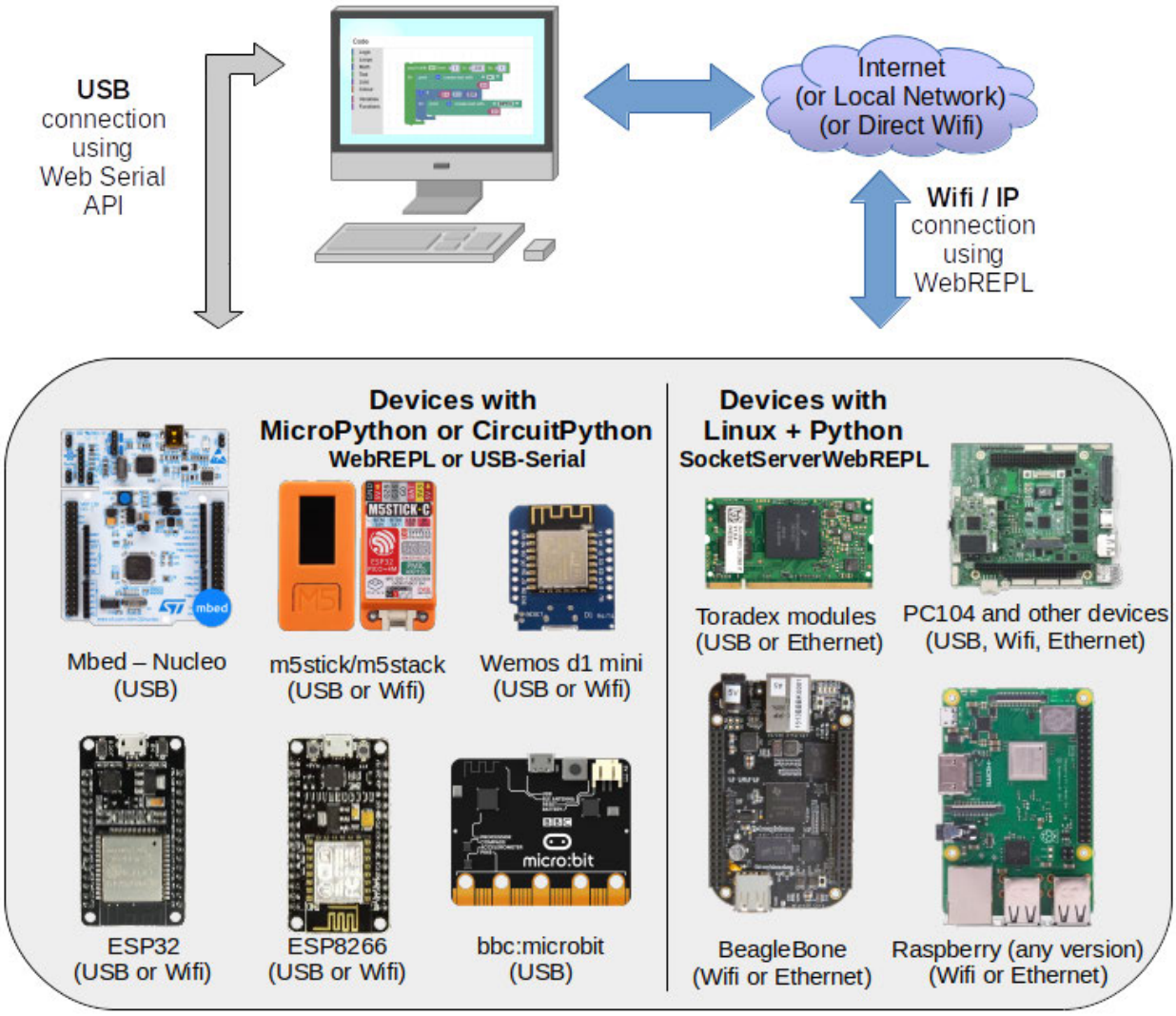

FIGURE 3. General overview of BIPES architecture and operation.

A typical embedded development scenario would require to write code, compile, flash the compiled code on the microcontroller and debug. With BIPES, there is no need to compile neither to flash the microcontroller. A single click transfers and executes the Python program. If needed, the program can be later saved in the flash memory filesystem for stand alone operation.

Figure 4 shows the main screen of BIPES when its web address (http://www.bipes.net.br/beta2/ui/) is accessed. As it can be seen, there are several tabs. In order to explain BIPES architecture, the function of each tab present in this home screen will be explained next.

A. BLOCKS

The *Blocks tab* contains all the blocks available for a certain device in a toolbox shown on the left side of the screen. Some blocks are native from Blockly, others were automatically or manually generated by BIPES team. The toolbox layout and block list is configured by a XML file, and the actions of each block are defined in additional files (generator_stubs.js and block_definitions.js).

Notice the ''Target device'' drop down selection box on the top right area of the browser. When the user selects a certain device from that drop down selection box, an event reloads the toolbox, showing only blocks related to functions available in that board.

Thanks to MicroPython and CircuitPython support for several hardware buses and sensors, BIPES takes advantage of these functions providing blocks for I2C, SPI, OneWire and UART Serial communication. Moreover, there are blocks for file operations and file system manipulation, which is a good advantage of classical microcontrollers. Blocks for several types of sensors and actuators are also provided.

In this way, custom blocks can be prepared for each board. For example, current BIPES version provides Timer blocks, which allow pseudo-parallel tasks execution in each Timer

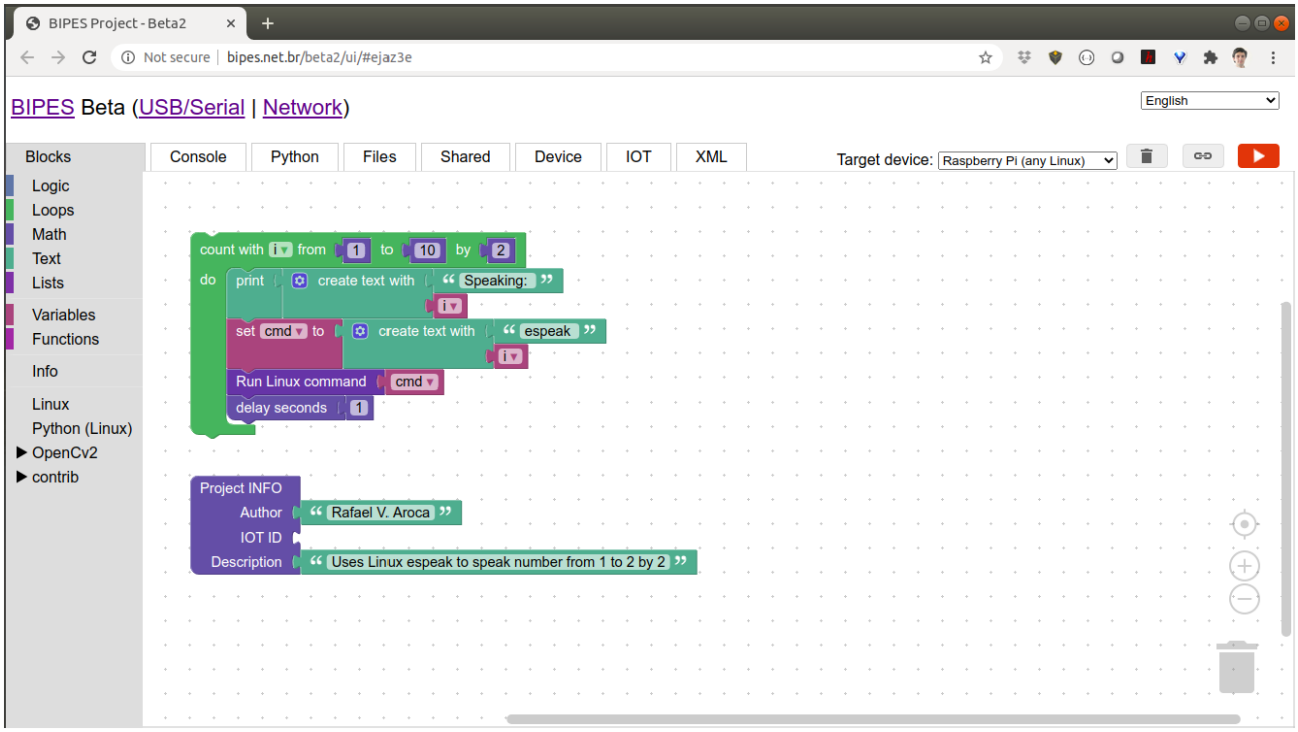

FIGURE 4. General overview of BIPES web based user interface and operation.

block, by Timer interrupts executing code associated with several hardware timers.

B. CONSOLE

The *Console tab* has an interactive console like any serial terminal software, as the PuTTY for example (https://www.putty.org/). It is based on term.js project by Christopher Jeffrey (https://github.com/chjj/term.js/) and integrated on BIPES with WebREPL and Web Serial API. In that way, it allows interactive REPL sessions from BIPES to any board, using USB or Network modes.

This console is, in fact, a key part of BIPES, so all communications from BIPES to the boards are done using this channel. Sending and receiving files, commands and running interactive Python / MicroPython / CircuitPython sessions.

Due to some web specification constraints, BIPES needs a workaround to use USB and WebSockets. USB communication from browser to devices is only permitted when the website is hosted in a secure (SSL / HTTPS) environment. WebSockets on MicroPython / CircuitPython devices, on the other hand, are only supported without SSL, so BIPES must connect to devices using standard WebSockets, which, by themselves, can only be established from non-SSL hosted pages. In this way, BIPES provides 2 addresses, one for USB connections and the other for networks connections. These options are available in the top left corner of BIPES screen, on the USB/Serial or Network links.

After selection of the type of connection, a user has simply to power on a board and connect to it, selecting its USB port or IP Address.

One powerful and practical feature of MicroPython and CircuitPython is that devices with Wifi connectivity can create Wifi Access Points when the board is powered on so that one user can simply connect to the board's network and interact with it using the Console tab of BIPES. If needed, for certain, a user can also configure a board to connect to an exiting Wifi network and connect to it using the existing infrastructure (Access Point) or simply connect by USB.

For devices with only USB connection, but with network connectivity requirement, we developed a protocol converter for BIPES named SerialWebSocketServer (https://github.com/rafaelaroca/SerialWebSocketServer). This tool establishes a bridge between a WebSocket and a USB (serial) connection, allowing BIPES WebREPL Console to access USB only devices connected to any computer or device capable of running Python. In such a way, this SerialWebSocketServer can be executed on Windows, Mac or Linux computers, or even on a BeagleBone or Raspberry Pi, providing network programming and control to a USB only board, such as the BBC micro:bit or the mBed Nucleo USB boards.

C. PYTHON

The *Python tab* shows the automatically generated code by BIPES, which can be also seen on the Files tab, and edited in Python (if desired) in that tab. Figure 5 shows an overview of the operation explained up to this point. Basically, a user designs its program using blocks, which are automatically converted into Python code and then sent to the board using USB or network. Standard output (prints) from programs are automatically redirected to the console, so the user can have

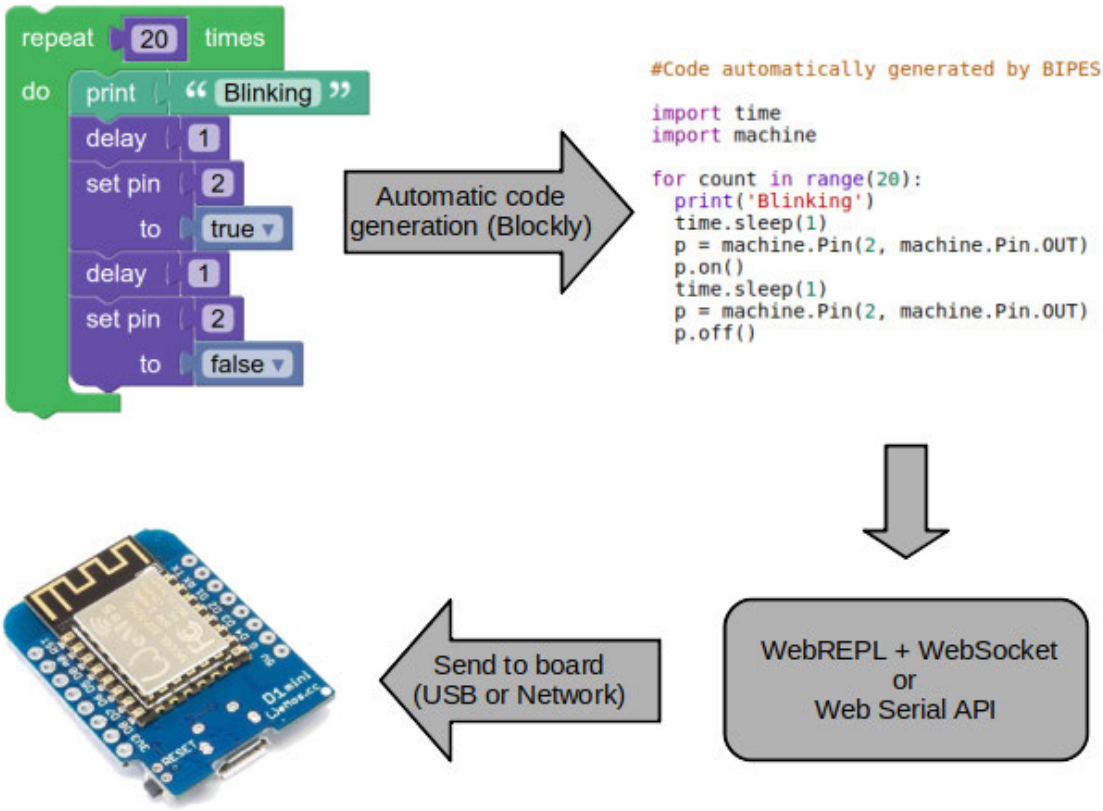

FIGURE 5. General overview of BIPES architecture and operation.

immediate feedback of the program operation. Note that the code generation, transmission and startup operation runs in a few milliseconds, allowing to perform quickly and simply the testing and debugging steps.

Notice that this presented approach is rather different than the one typically used with traditional microcontroller programming, which usually involves uploading binary codes/sketches to the board, which are written in the device's flash memory and executed after the board reboots. This can also be done using bootloader softwares, but the process of flash writing is still needed, which takes several seconds or even minutes, making it more difficult to quickly debug / adjust programs.

D. FILES

One interesting and powerful feature supported by MicroPython and CircuitPython is the implementation of a file system in the device's flash memory. With this feaature, program, libraries, configuration and data can be directly stored in the devices' flash memory as files.

The *Files tab* provides additional functions to manipulate files in the mentioned file system using the WebREPL protocol. Users can send local files directly to the device, which will be permanently stored in the device's file system. Files can also be downloaded from the board, or opened in a Python text source code editor in this same tab.

The text editor allows users to directly program and test Python code on the target board. To do this, several approaches are possible. First one is to view and edit the Python code automatically generated by the blocks, which is interesting for educational purposes or fine tuning generated code. A second option is to directly type Python code in the Files tab. Any code typed or generated by blocks can be saved and executed using the buttons Save, Run and Stop.

One last feature is to load Python code templates or libraries from BIPES server. This is useful for situations where some device needs drivers or specific libraries. Therefore, the library can be provided by a server and automatically loaded in the Python editor and sent / written to the board's flash memory. One example is the OLED LCD Display, SSD1306, which requires some functions from the ssd1306.py library, which is not provided with MicroPython or CircuitPython firmware. Another related possibility provided by BIPES is the automatic installation of additional libraries needed by specific blocks, by simply clicking on the ''Install library'' button on the blocks toolbox.

E. SHARED

The *Shared tab* offers a quick and simple way to open examples and block programs shared by the BIPES community. Whenever someone wants to share his code with others, he just needs to click on the link button on the top right area of BIPES interface (at the left of the run button). After clicking this button, a script will save the current blocks on the screen, send it to a server and define a unique link to access this program (blocks) again by simply clicking on the link. With

this, it is possible to share the link with anyone interested in viewing, discussing, or even using a program. Such one example of a link is at http://bipes.net.br/beta2/ui/#npoksg.

In order to keep the shared files organized, the blocks will only be saved and the link generated if the block *Info: Project Info* is used. This block allows programmers to provide a description for the block based program, author name and an ID for IoT applications. Such information is shown on the list that can be seen on the Shared tab. Clicking in one of the options on the shared tab will open the shared block based program.

F. DEVICE

The *Device tab* is an area to provide custom documentation for each board that can be programmed with BIPES. As it happens with the toolbox, when a different target device is selected, this tab is automatically updated to show the pinout diagram of the selected device and other useful information for that specific device. The idea is to provide a quick reference for designers, and links to more detailed information.

G. IoT

The *IOT tab* provides web access to a simple database of sensor data. In this way, any device connected to the Internet can send data to this IOT backend by using HTTP GET requests. Sensor and other information are sent as part of the URL. For example: http://www.bipes.net.br/sensors/send.php?id = 90&s1 = 7.

From the example URL, data sent to the server will be stored associated with the identification (ID) 90, and the column for sensor 1 (s1) will store the number 7. Several sensors can be stored in the same request, by adding other variables to the URL. Each request to the server will store the provided data, with an associated timestamp provided by the server at the instant of data reception.

With the data sent to the server, it is possible to view such data in a graph (single or multi-line), with configurable parameters, such as line names, date and time interval. It is also possible to view a real time graph, with automatic updates of the data sent to the server or to view a table with real time updates. If needed, a comma separated value (CSV) can be downloaded for offline analysis using other tools, such as Excel or LibreOffice Calc spreadsheets.

H. XML, INTERNATIONALIZATION AND SECUTIRY

The *XML tab* is a simple view of the XML generated by the blocks. Another relevant feature is multi-language support. With this, commands can be translated to other languages, for example, Portuguese (somewhere tested) or Spanish. With this, the BIPES can be even more inclusive and used in countries in their own languages.

A final notice is about security. It should be clear that the presented scheme offers no authentication, privacy or security of the data sent to the server. It is provided as resource to enable quick and easy start and testing IoT and data sharing. Professional applications should consider privacy, secure protocols, authentication and other security issues according to the requirements of each project.

I. TARGET DEVICES WITH PYTHON

The approach provided by BIPES to program low cost microcontrollers can be also extended to more powerful devices. In order to accomplish such task, a Python REPL server using WebSockets was also developed: SocketServerWebREPL.[1] This software can be executed in any Python environment, such as a standard computer running Python, a Raspberry Pi or a BeagleBone board running Python. Such setup is shown on Figure 6.

The setup allows advanced features and blocks to be included to BIPES. Not only by providing access to a full version of Python and libraries, but also by enabling use of other resources, programs and features of these powerful platforms through Python integration.

One example of integration, already available as a BIPES Block is to execute shell commands on a Linux or Windows device. In that way, the block based program can invoke external commands, programs and tools according to certain conditions. For example, a block based program can invoke an external voice synthesis program to allow the system to ''speak''. This takes all the advantage of one Unix philosophy, as written by Salus [21]: ''Write programs to work together''. In this same direction, as many Unix Shell scripts that can do a variety of tasks by invoking other programs, BIPES can also take advantage of such feature.

Another feature already implemented on the current BIPES version is computer vision blocks using OpenCV, allowing advanced computer vision programs to be developed, deployed and tested using blocks. Some examples are shown in section V. OpenCV blocks were automatically generated using an open source tool that parses OpenCV source code and generate Blockly files (https://github.com/berak/blockly-cv2/). Several adjusts and interactions had to be done. However, the automatic code generation of OpenCV blocks was quite useful.

Finally, another important feature of BIPES, as a web platform, is that when new boards are added to BIPES main site, all users can automatically start using these new boards, features, or libraries. In the traditional way, libraries should be downloaded and installed, or new development IDEs should be downloaded and installed for new boards.

V. BIPES BETA (CURRENT STAGE OF DEVELOPMENT) AND EXAMPLES

We started BIPES development in Mach 2020, in such a way that the current version can still considered a beta version. However, it is already usable and fully functional, as the examples shown next.

Figure 7 shows a complete example of a simple temperature control loop implemented on a board based on a ESP8266 or ESP32 board. An analog temperature sensor

[1] https://github.com/rafaelaroca/SocketServerWebREPL

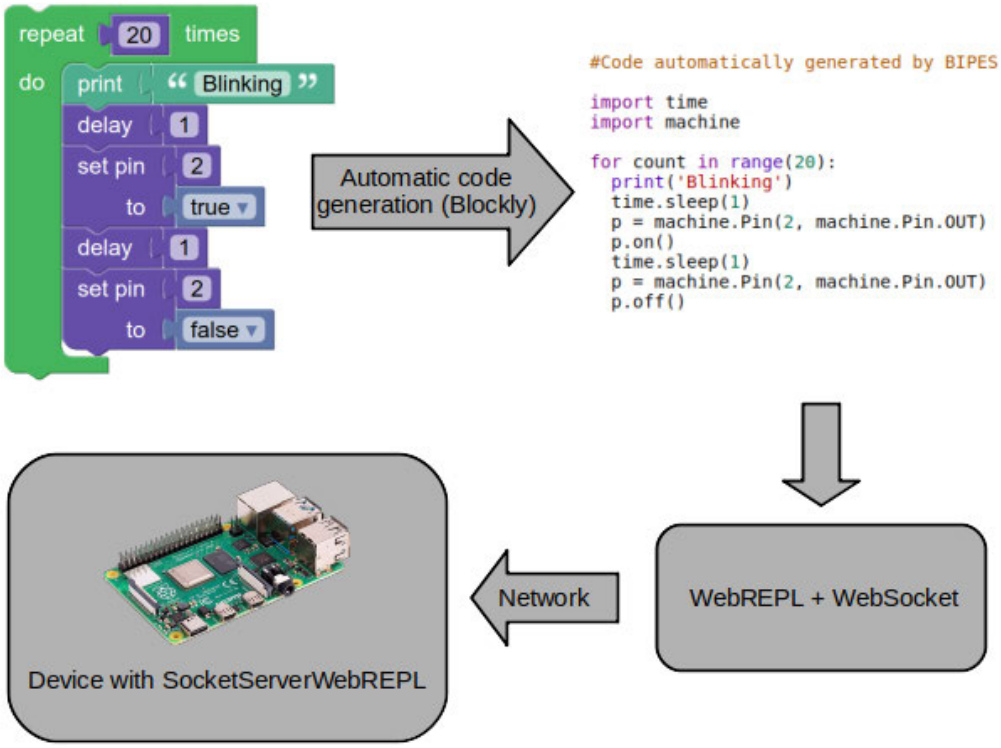

FIGURE 6. Overview of BIPES usage with Python enabled devices and using SerialWebSocketServer.

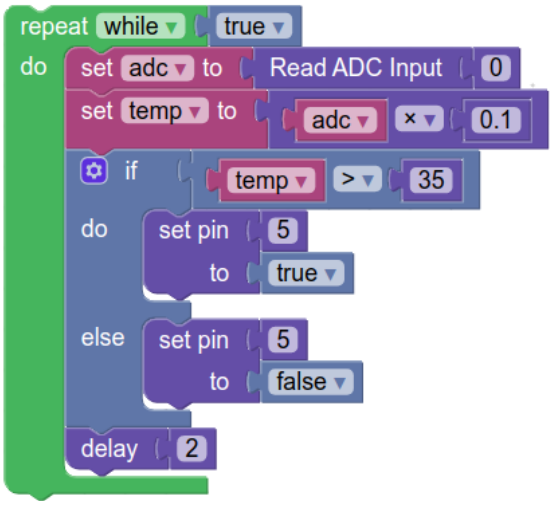

FIGURE 7. Complete simple control loop example.

is connected to the board's analog pin, which is read and multiplied by 0.1 for scale conversion. Next, the program check if the temperature is higher than 35 degrees, and turns a fan on or off using a relay, to control the temperature. The process sleeps for 2 seconds and restarts (repeat while true block).

As seen above, Figure 2 has shown a complete IoT example, which is an analog sensor connected to the analog input of a ESP32 or ESP8266 board is read. The reading is stored in a variable local to the program (value) and joined with another variable, which is a HTTP GET request to store the reading on BIPES server. Next, HTTP GET block sends the data to the server using HTTP GET method. The system then sleeps for 5 seconds and repeats the operation more 99 times. The result can be seen in the IOT tab of BIPES. Figure 8 shows an image of a historic graph generated from the data sent using the block program from figure 2. Such graph can be seen / generated from BIPES IOT tab, where a user can specify figure captions, date and time range, and view the graph (one line or multi-line). Real time graph is also available in the IOT tab.

Figure 9 shows a block program prepared with BIPES and executed on a Raspberry Pi running SocketServerWebREPL. Such program guesses numbers from 1 to 100 and speaks the guessed numbers using the speakers of the Raspberry Pi. This is done by calling the espeak text-to-speech Linux utility.

Computer vision applications could also be developed using BIPES blocks and generating Python code with the OpenCV Python library. Figure 10 shows a simple OpenCV application, which crops a region of interest (ROI) of the image and shows both images live, from the camera onto the monitor. The target device that can be a Raspberry Pi, Beagle Board or an x86 PC.

Figure 11 shows another OpenCV application, a motion detector. The blocks create a program that captures live images from a camera, and subtracts each frame from the

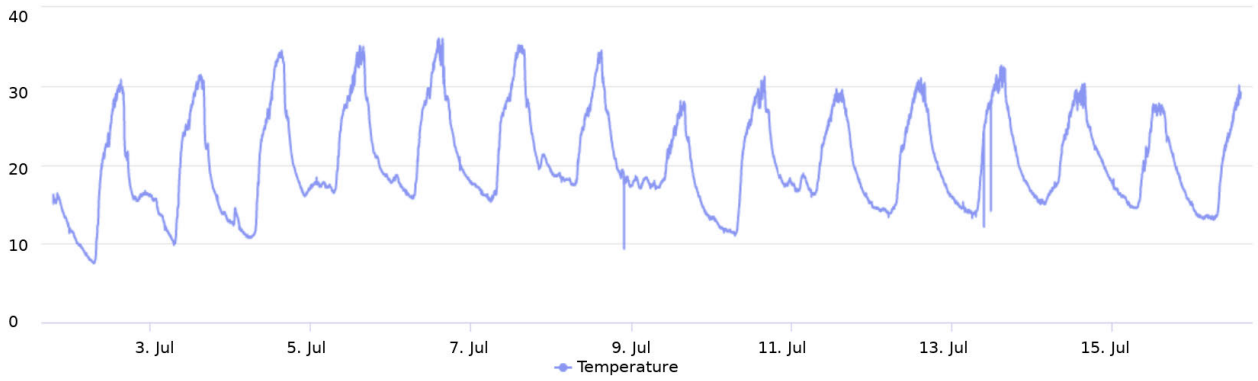

FIGURE 8. Example of temperatures during 15 days, from a real temperature sensor and collected / stored using BIPES.

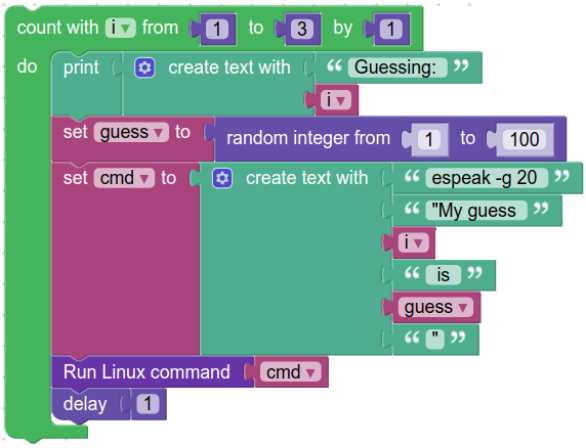

FIGURE 9. BIPES guessing numbers and speaking them through a Raspberry Pi using Linux.

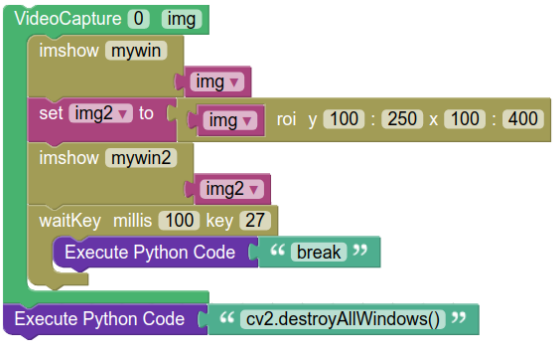

FIGURE 10. OpenCV integration with BIPES.

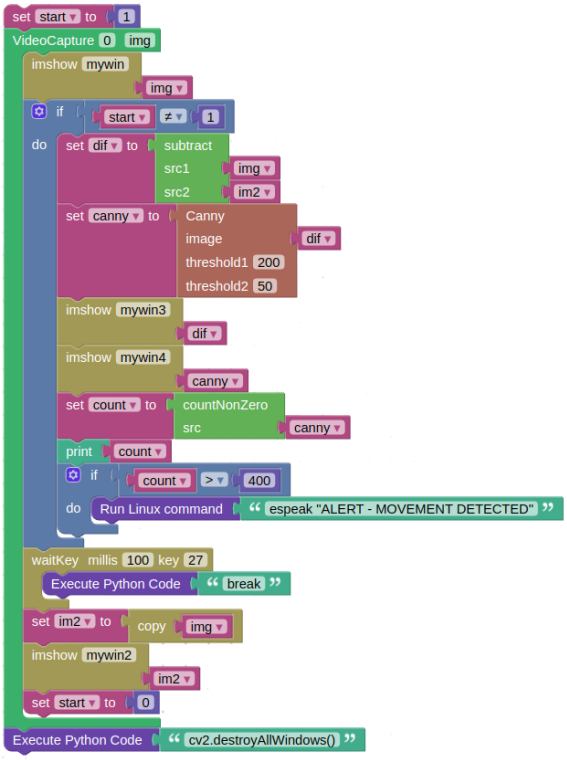

FIGURE 11. OpenCV movement detection code.

frame captured 100 milliseconds ago resulting in a difference image (dif. variable). Next, a Canny edge detector is applied and the number of non-zero pixels is counted. If there is no movement, this count is very low. A threshold of 400 is used, and if the non-zero pixels is larger than 400, an action is triggered. In this example, the system speaks "ALERT - Movement Detected". Notice that other actions could be easily integrated, such as sending information to a server or activating a general purpose input output (GPIO) port to turn on a light, for example. Figure 12 shows the result of the execution of these blocks, where we can see the last live image, the frames subtraction and the border from the Canny operator.

Another point of flexibility of BIPES is to install new modules and use them from a block program. Figure 13 shows a BIPES program with a simple Artificial Intelligence example using the scikit-learn package for machine learning with Python. The example is a block version from the first scikit-learn examples (Fitting and prediction: estimator basics). The program itself installs the needed dependencies

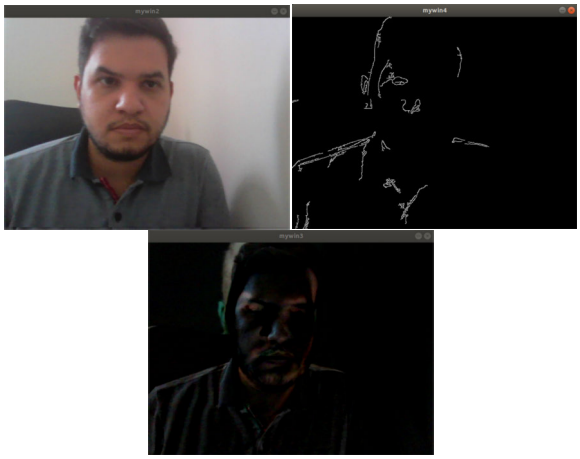

FIGURE 12. Output from the OpenCV program to detect movement and trigger an event created with the program of Figure 11.

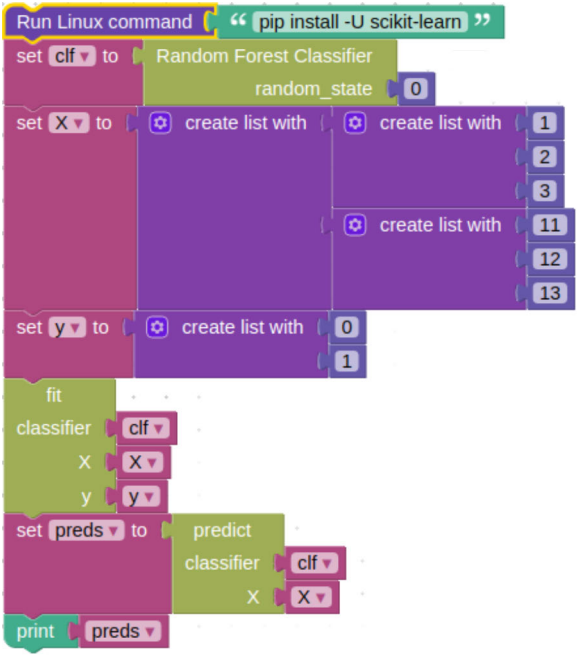

FIGURE 13. Basic classifier algorithm using scikit-learn.

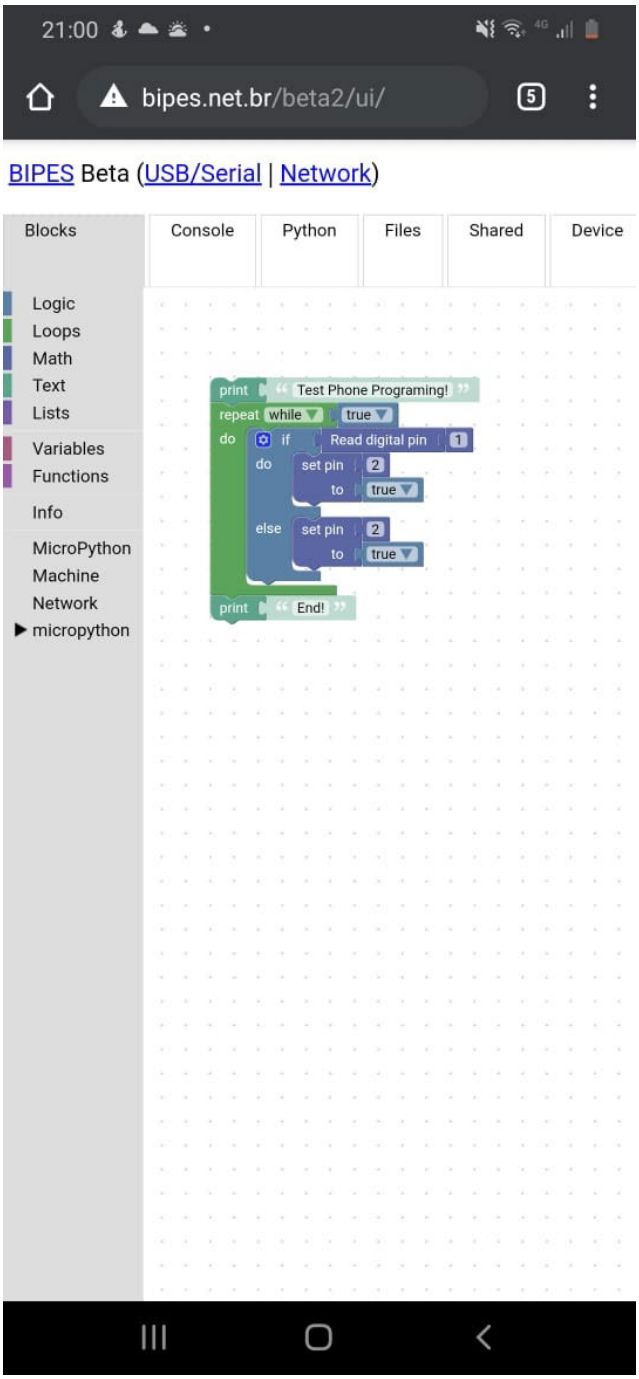

FIGURE 14. BIPES being used from an android smartphone.

and then creates a simple classifier where X are input samples and Y is the output. After training, the Fit function learns the input pattern and gives output as answers.

As for the BIPES web interface, it has been tested in Android devices (tablet and smartphones), Linux and Windows machines with Google Chrome and Firefox, iOS devices from Apple, Amazon Fire Stick and even PlayStation 4 (PS4) gaming console, where all features worked using PS4 default web browser. Figure 14 shows a capture of BIPES being used on an Android smartphone.

A video demonstration showing an overview about BIPES functions can be seen at youtube (https://www.youtube.com/watch?v=BqGes0sZPM4). Between the several functions, the demo includes program creation, editing and saving, connection, live data and historic data viewing, device information viewing, and program sharing besides blocks customization per device, and sending through the USB or network.

VI. CONCLUSION

Embedded systems are pervasive in any current application environments, as in robotics, heating control, smart homes, printers, media devices, computers, cars, airplanes, health

care, industry, toys, and much more. Besides, low cost computing devices with high processing capacity and features are each time more present and accessible in several scenarios. More than that, such devices can foster more innovations and developments for Internet of Things applications, education, and other custom products and devices. In this way, it is important to be able to quickly program, test and interact with such devices. Moreover, learning programming offers several advantages and good results in skills development [1].

BIPES has been proposed to deal with these issues, integrating several technologies, providing an intuitive and quick approach to learn and develop solutions using embedded processors, specially for starters and new users. In fact, starters can have a faster learning curve as BIPES does not need any software installation or configuration. Moreover, as algorithms are updates and new libraries added to BIPES, they are automatically available to all users. As BIPES supports several target platforms, new users can also have easy access to devices compatible with BIPES. Besides, it is also a powerful tool for experienced users, as it allows direct Python programming over the network, and a quick approach to transfer files, programs and debug applications on several computing platforms.

The focus of this paper is to introduce BIPES architecture. One of its main features is to be fully based in standard HTML5 technologies, so no software or plugin installation is needed, allowing BIPES to be used (as a programming platform) on several types of devices, from computers and smartphones to video-games. As for target devices, BIPES generates Python code that can be executed in a wide variety of devices, including low cost modules, such as ESP8266 to high performance x86 computers.

It is already fully functional, but as other open source tools, it is being constantly updated and improved. Moreover, it can take advantage of all Python ecosystem. For example, BIPES already has OpenCV blocks integrated to allow computer vision applications to be developed using BIPES on Linux based platforms.

To validate BIPES, the paper presented several success cases of programs created with it, quickly and in an intuitive way. We hope that BIPES can foster several teaching activities and also the development of new products and services. BIPES is an Open Source software, with its source code shared in github (repository info at http://www.bipes.net.br), in the terms of the General Public Licence (GPL), so contributions and improvements are welcome. BIPES is also available as a free programming platform, so any user can have access at its web site (http://www.bipes.net.br), and start programming embedded systems using BIPES' blocks.

BIPES is already fully functional, but there is a road-map for a bunch of new functionalities that can be developed by BIPES team or by the community. Internationalization is one of them, as other countries users can translate BIPES and make it available to several languages. Also, another way to improve is to create more blocks, including several types of sensors and device features. Nonetheless, the creation of blocks for integration with other tools, such as MIT App Inventor, Blynk, IFTTT and ThingSpeak is another possibility. Also, including a more responsive user interface (UI) is one of our goals for improving BIPES, and to configure it for more boards and devices. In special, to connect it with portable and wearable devices as gloves [29], making it possible and easy to program them ''on the field''.

Several other ideas for future enhancements are to have a multi-language documentation about BIPES, boards, video tutorials and more detailed instructions, including the development and publication of block program examples. Auxiliary tools as bugfixes in BIPES, SocketServerWebREPL and other related tools, are also example of missing capabilities, including the security issue mentioned above. So to provide authentication and security mechanisms for BIPES ecosystem will also be included in our future works, plus a cloud compilation server in order to generate compiled bytecode from BIPES blocks and load these codes in boards already available and boards less powerful, such as Arduinos.

ACKNOWLEDGMENT

The authors would like to thank the institutions for providing infrastructure and support. Finally, the authors would like to thank several authors of all Open Source tools and technologies used by BIPES Development Team.

REFERENCES

[1] S. Hodges, S. Sentance, J. Finney, and T. Ball, ''Physical computing: A key element of modern computer science education,'' *Computer*, vol. 53, no. 4, pp. 20–30, Apr. 2020.

[2] D. Weintrop, ''Block-based programming in computer science education,'' *Commun. ACM*, vol. 62, no. 8, pp. 22–25, Jul. 2019, doi: 10.1145/3341221.

[3] B. Jansen and F. Hermans, ''XLBlocks: A block-based formula editor for spreadsheet formulas,'' in *Proc. IEEE Symp. Vis. Lang. Hum.-Centric Comput. (VL/HCC)*, Oct. 2019, pp. 55–63.

[4] D. Weintrop, A. Afzal, J. Salac, P. Francis, B. Li, D. C. Shepherd, and D. Franklin, ''Evaluating CoBlox: A comparative study of robotics programming environments for adult novices,'' in *Proc. CHI Conf. Hum. Factors Comput. Syst. CHI*, 2018, pp. 1–12, doi: 10.1145/3173574.3173940.

[5] J. M. Rodriguez Corral, I. Ruiz-Rube, A. Civit Balcells, J. M. Mota-Macias, A. Morgado-Estevez, and J. M. Dodero, ''A study on the suitability of visual languages for non-expert robot programmers,'' *IEEE Access*, vol. 7, pp. 17535–17550, 2019, doi: 10.1109/ACCESS.2019.2895913.

[6] R. D. Caballar, Programming Without Code: The Rise of No-Code Software Development. IEEE Spectr. Tech Talks. Accessed: Nov. 2 2020. [Online]. Available: https://spectrum.ieee.org/tech-talk/

[7] W. Wolf, ''The embedded systems landscape,'' *Computer*, vol. 40, no. 10, pp. 29–31, 2007.

[8] J. Lee, G. Park, J. Shin, J. Lee, C. J. Sreenan, and S. Yoo, ''SoEasy: A software framework for easy HardwareControl programming for diverse IoT platforms,'' *Sensors*, vol. 18, no. 7, p. 2162, 2018, doi: 10.3390/s1807216.

[9] J. Á. Ariza, ''DSCBlocks: An open-source platform for learning embedded systems based on algorithm visualizations and digital signal controllers,'' *Electronics*, vol. 8, no. 2, p. 228, Feb. 2019, doi: 10.3390/electronics8020228.

[10] W. Z. Khan, M. H. Rehman, H. M. Zangoti, M. K. Afzal, N. Armi, and K. Salah, ''Industrial Internet of Things: Recent advances, enabling technologies and open challenges,'' *Comput. Electr. Eng.*, vol. 81, Jan. 2020, Art. no. 106522.

[11] J. Devine, J. Finney, P. de Halleux, M. Moskal, T. Ball, and S. Hodges, ''MakeCode and CODAL: Intuitive and efficient embedded systems programming for education,'' *J. Syst. Archit.*, vol. 98, pp. 468–483, Sep. 2019.

[12] M. Hauck, R. Machhamer, L. Czenkusch, K.-U. Gollmer, and G. Dartmann, "Node and block-based development tools for distributed systems with AI applications," *IEEE Access*. vol. 7, pp. 143109–143119, 2019.
[13] M. Khamphroo, N. Kwankeo, K. Kaemarungsi, and K. Fukawa, "MicroPython-based educational mobile robot for computer coding learning," in *Proc. 8th Int. Conf. Inf. Commun. Technol. Embedded Syst. (IC-ICTES)*, May 2017, pp. 1–6.
[14] M. Khamphroo, N. Kwankeo, K. Kaemarungsi, and K. Fukawa, "Integrating MicroPython-based educational mobile robot with wireless network," in *Proc. 9th Int. Conf. Inf. Technol. Electr. Eng. (ICITEE)*, Oct. 2017, pp. 1–6.
[15] W. Xin and E. W. Patton, "A blocks-based approach to Internet of Things in MIT App inventor," in *Proc. BLOCKS+ Workshop, Co-Located SPLASH*. Nov. 2018, pp. 1–2.
[16] E. Pasternak, R. Fenichel, and A. N. Marshall, "Tips for creating a block language with blockly," in *Proc. IEEE Blocks Beyond Workshop (B&B)*, Oct. 2017, pp. 21–24.
[17] N. Fraser, "Ten things we've learned from blockly," in *Proc. IEEE Blocks Beyond Workshop (Blocks Beyond)*, Oct. 2015, pp. 49–50.
[18] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin, "Blockly goes to work: Block-based programming for industrial robots," in *Proc. IEEE Blocks Beyond Workshop (B&B)*, Oct. 2017, pp. 29–36.
[19] K. Prutz and H. Abelson, "Expanding device functionality for the MIT App inventor IoT embedded companion," Student Term Paper, Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep. 12, 2018.
[20] D. A. Hume, *Progressive Web Apps*, 1st ed. New York, NY, USA: Manning Publications, 2017.
[21] P. H. Salus, *A Quarter Century of UNIX*. Reading, MA, USA: Addison-Wesley, 1994.
[22] H. Liao, J. Jiang, and Y. Zhang, "A study of automatic code generation," in *Proc. Int. Conf. Comput. Inf. Sci.*, Dec. 2010, pp. 689–691, doi: 10.1109/ICCIS.2010.171.
[23] Q. Zhuang, J. Feng, and H. Bao, "Measuring semantic gap: An information quantity perspective," in *Proc. 5th IEEE Int. Conf. Ind. Informat.*, Jun. 2007, pp. 669–674.
[24] T. Ball, A. Chatra, P. de Halleux, S. Hodges, M. Moskal, and J. Russell, "Microsoft MakeCode: Embedded programming for education, in blocks and TypeScript," in *Proc. ACM SIGPLAN Symp. SPLASH-E SPLASH-E*, Oct. 2019, pp. 7–12.
[25] N. K. Singh, "EB2ALL: An automatic code generation tool,"in *Using Event-B for Critical Device Software Systems*. London, U.K.: Springer, 2013, doi: 10.1007/978-1-4471-5260-6_7.
[26] D. Pilaud, "Efficient automatic code generation for embedded systems," *Microprocessors Microsyst.*, vol. 20, no. 8, pp. 501–504, Apr. 1997.
[27] *McConnell, Steve. Code Complete*. London, U.K.: Pearson, 2004.
[28] D. Fiehler, B. Collins, and J. Carlaftes. (Oct. 2009). *Using Model-driven Engineering Techniques for Integrated Flight Simulation Development*. Raytheon. On-Line. [Online]. Available: https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2009/systemengr/8980ThursdayTrack4Fiehler.pdf
[29] R. Aroca, R. Gomes, R. Dantas, A. Calbo, and L. Gonçalves, "A wearable mobile sensor platform to assist fruit grading," *Sensors*, vol. 13, no. 5, pp. 6109–6140, May 2013.

ANDOUGLAS GONÇALVES DA SILVA JUNIOR received the degree and M.Sc. degree in mechatronics engineering from the Federal University of Rio Grande do Norte, where he is currently pursuing the Ph.D. degree in electrical and computer engineer with a sandwich period in ISASI Lab, Italy. His research interests are machine learning, robotics, and digital holography.

LUIZ MARCOS GARCIA GONÇALVES (Member, IEEE) received the Ph.D. degree in systems and computer engineering from the Federal University of Rio de Janeiro, in 1999. He is currently a Full Professor with the Federal University of Rio Grande do Norte, Brazil. His research interests include graphics processing with applications to computer vision and robotics fields and robotics in education. He was the Founder and Chair of the Brazilian Committee on Robotics and the Chair of the Computer Graphics and Image Processing Committee, both under the Brazilian Computer Society.

GLAUCO A. DE PAULA CAURIN (Member, IEEE) has a specialization in mechatronics and received the Ph.D. degree from the Swiss Federal Institute of Technology - ETH, Zurich, Switzerland, in 1990 and 1994, respectively. He was a Mechanical Engineer with the EESC – USP Brazil, in 1988. From 2010 to 2011, he held a sabbatical year at the Newman Laboratory for Biomechanics and Human Rehabilitation, MIT, Cambridge, MA, USA. He is currently a Full Professor with the Department of Aeronautical Engineering, EESC - USP. His interests include autonomous systems, UAVs, real-time embedded systems, neural networks, collaborative robotics, surgery, and rehabilitation robots.

GUSTAVO TERUO BERNARDINO TAMANAKA received the bachelor's degree in mechanical engineering from the Federal University of São Carlos, in 2017, where he is currently pursuing the master's degree in mechanical engineering. He is also a partner at SFLabs, a tech startup in São Carlos, with a specialization in areas like computer vision, web systems, and embedded systems.

ANDRÉ CARMONA HERNANDES received a degree in mechatronics engineering, and the M.Sc. and Ph.D. degrees from the University of São Paulo. He has about ten years of experience in mobile robotics. His main research interests are aerial robotics and embedded systems.

RAFAEL VIDAL AROCA (Senior Member, IEEE) received a degree in informatics and the M.Sc. degree in mechatronics engineering from the University of São Paulo, and the Ph.D. degree in electrical and computing engineering. He has over ten years of industry experience in embedded systems, IT systems, and servers administration. He is currently an Adjunct Professor with the Federal University of São Carlos. His main research interests are in embedded systems, operating systems, and robotics.