electronics

MDPI

Article

FOSSBot: An Open Source and Open Design Educational Robot

Christos Chronis † and Iraklis Varlamis *,†

Department of Informatics and Telematics, Harokopio University of Athens, Omirou 9, 17778 Athens, Greece; chronis@hua.gr

* Correspondence: varlamis@hua.gr; Tel.: +30-210-9549405

† Current address: Omirou 9, 17778 Athens, Greece.

Abstract: In the last few years, the interest in the use of robots in STEM education has risen. However, their main drawback is the high cost, which makes it almost impossible for schools to have one robot per student. Another drawback is the proprietary nature of commercial solutions, which limits the ability to expand or adapt the robot to educational needs. Different robot kit versions, which have different electronics and programming interfaces and target different age groups, make the decision of educators on which robot to use in STEM education even more complicated. In this work, we propose a new low-cost 3D-printable and unified software-based solution that can cover the needs of all age groups, from kindergarten children to university students. The solution is driven by open source and open hardware ideas, with which, we believe we will help educators in their work. We provide detail on the 3D-printable robot parts and its list of electronics that allow for a wide range of educational activities to be supported, and explain its flexible software stack that supports four different operating modes. The modes cover the needs of users that do not know or want to program the robot, users that prefer block-based programming and less or more experienced programmers who want to take full control of the robot. The robot implements the principles of continuous integration and deployment and allows for easy updates to the latest software version through its web-based administration panel. Though, in its first steps of development and testing, the proposed robot has a huge potential, due to its open nature and the community of students, researchers and educators, that potential has kept growing. A pilot at selected schools, a performance evaluation of various technical aspects and a comparison with state-of-the-art platforms will soon follow.

Keywords: STEM; educational robotics; 3D-printable robot; Blockly; open design; open software

Citation: Chronis, C.; Varlamis, I. FOSSBot: An Open Source and Open Design Educational Robot. *Electronics* **2022**, *11*, 2606. https://doi.org/10.3390/electronics11162606

Academic Editor: Ahmad Taher Azar

Received: 25 July 2022
Accepted: 19 August 2022
Published: 20 August 2022

1. Introduction

The use of science, technology, engineering and mathematics (STEM) tools has proven to be valuable for both teachers and students in various educational contexts. Robotics has a big impact on STEM education, since it offers a constructive learning environment that is ideal for explaining, simulating and teaching both scientific and non-scientific subjects [1]. It can be particularly helpful in educating STEM students because it allows engineering and technology principles to be applied in real world cases, reducing the abstractness of science and mathematics. In this direction, numerous robotics-related activities enhance students' understanding of science, technology, engineering and/or mathematics, and stimulate the interest of students in other non-scientific fields, such as art, history or literature [2–4].

Studies of the usage of STEM in early childhood [5] and on the status and trends in STEM education research [6] summarize the majority of the latest publications in this field and discuss a number of issues concerning its increased popularity. Based on the aforementioned publications, STEM tools are considered to be important and innovative and can shape education from kindergarten to university in a wide variety of fields [7,8].

The core idea behind STEM is the constructionism learning theory developed by Papert [9]. Based on this theory, the students interact with tools and materials, use previous knowledge and experiences and construct new information [2]. The STEM tools offer these

means and provide the opportunity to students to solve real-world problems and, at the same time, to expand their personal experiences, as they become active learners [2,3,7,10]. The interaction with physical devices can transform the procedure of learning into a fun activity, and can keep the students' interest in learning at a high level [11]. Through this process, students have the opportunity to improve their critical thinking and problem-solving skills [12,13]. Many studies also report that it can also improve collaboration, social and communication skills, cognitive, meta-cognitive and social responsibilities [12,14] through the improvement of self-confidence and self-direction [7,12,13,15]. Finally, the exposure to the use of the STEM tools can also give students the needed inspiration for their future careers, as was highlighted in [16].

Nowadays, the skills of algorithmic thought and programming are becoming fundamental knowledge blocks, since almost all of the science fields require such competencies. Especially in the last decade, the continued rise of machine learning (ML) and artificial intelligence (AI) applications has put AI-based systems in the front line of most companies. The programming, aside from the potential benefits of a future job in education, can be used to teach computational thinking and problem-solving competence. There have been many approaches to teaching programming [17] but, for many students, it still remains a difficult and, in some cases, boring task to perform [4,17]. That difficulty is due to the fact that a student has to understand the theory of algorithms thought abstract concepts, or to learn the syntax and the commands of a programming language by memorizing or testing them in a virtual scenario. An additional barrier is raised for students that do not speak English as their native language, since most of the programming languages use English keywords to represent syntactic and semantic rules. All of the previous reasons and difficulties can lead the students to adopt a negative attitude towards programming [18].

Much research has been carried out in order to determine how the STEM tools (robotic kits) can positively affect the education procedure [19,20]. The usage of educational robots is a good practice for an educator in order to tackle the previous problems. A robot allows learners to establish new and creative ways to solve real-world problems, to test their ideas and to experiment. A unique ability provided by robotic systems is the direct execution of a solution and the direct feedback from a decision, either right or wrong. Furthermore, it is more interactive and tangible than a simulation environment [21]. This interaction between the robot and the student creates a unique feeling to a student that accomplishes a task, and forms an excellent playground for testing and building his/her puzzle-solving skills [4,18,19].

At the same time, a real-life object can give the needed motivation to a student to utilize any previous knowledge and try to find optimal solutions that can transform a boring procedure into a game. An educator selecting the correct robot for their students can reach students of different ages, different intellectual backgrounds or with learning disorders in order to access learning [22–24]. In addition, many researchers suggested that the usage of a game with fantasy or strategy and some goals can transform a game into an educational tool [19,25,26].

A very important role of the acceptance of a robotic system in education is the user interface or, more precisely, the way that a student can interact with it and transfer their solution into the robot commands. The user interface has been widely studied with different robotics kits in many group ages [4].

In this article, we present the Free and Open Source Software Bot (FOSSBot (https://github.com/eellak/fossbot (accessed on 10 August 2022)) for short), an open source and open design robot that can be used for educational purposes at all levels of the educational system, supporting many different activities and teaching scenarios. The unique features of FOSSBot can be summarized in the following:

- The designs of the robot are open and can be modified accordingly in order to support more scenarios.
- The plastic parts of the robot can be printed in a 3D-printer at a very low cost. Any worn plastic parts can be easily 3D-printed again

- FOSSBot comprises a list of electronics that can be easily found in the market at a low cost. All parts can be assembled following the assembly instructions, which are also publicly available. Any faulty electronic parts can be easily replaced.
- The software stack, which is written in Python, is modular and containerized, and thus can be easily expanded with more functionalities, allowing the robot users to update to the latest version at the click of a button.
- The robot is self-sustained and remote-operated, which means that anyone can connect to it through a laptop, smartphone or tablet, or it can connect to the local network via WiFi.
- The programming of the robot can be carried out in a multitude of ways, which makes it accessible and usable by different ages and in different contexts: (i) directly in Python at the lowest level, (ii) using interactive notebooks (Python Jupyter) at the middle level, (iii) using a no-code interface (Google Blockly) or (iv) using a button-based UI that executes pre-defined code scripts.

Section 2. which follows, surveys the main work in the field. Section 3 introduces FOSSBot and provides details on its main hardware features, whereas Section 4 explains FOSSBot's software stack and its main operation modes. Section 5 briefly presents how FOSSBot is designed to fit the educational scenarios at different age levels and Section 6 concludes the paper with our next steps on improving the robot functionalities and on exploiting FOSSBot in various educational activities.

2. Related Work

In the last two decades, tangible programming has seen a great development. From the Logo language that was designed in the 1970s by Feurzeig and Lukas for teaching mathematics [27] and AlgoBlock [28] that was introduced in the 1990s by Suzuki and Kato for teaching algorithms, using a programming language with interlocking blocks to represent commands, we have moved to the Tangible Programming Bricks of McNerney in the 2000s [29], which support the use of parameters and variables.

However, the real boost to robot programming has come from LEGO during the last decade. The Bricks interface was used by LEGO bricks combined with embedded electronics through the Logo Block and LEGO Mindstorms [4,18,20]. More specifically, LEGO Mindstorms combined hardware blocks with the graphical representation of the code that operates them, thus giving a boost to tangible programming and converting it into a game-like activity. Every LEGO kit includes a big brick with a microprocessor inside and, around it, everyone can attach different kind of bricks, such as motor bricks and modular sensors. In addition, besides the hardware, LEGO provides a graphical programming interface. In that interface, a user has the opportunity to create complicated programming using a graphical representation of the bricks, and can then manipulate the robot [10].

The popularity of Mindstorms gave rise to more programming interfaces, such as FlowBlock [25]. FlowBlock used an arrows-to-blocks and a branches-based visual representation of the programming flows with real-time value updates. Similarly, authors in [30] proposed graphical representations of commands and, in [31], the authors introduced Electronic Blocks, which are tangible programming elements for preschoolers that allowed students to build and program a robot. Finally, authors in [32,33] proposed the Quetzal and Tern languages, which emphasize the use of cheap and durable components that do not contain any electronics but can be used offline to compile a program. The blocks are scanned and the resulting program is compiled accordingly in a smartphone or tablet.

Moving from components that can be assembled to programmable robots, we still find several interesting approaches. The great acceptance from educators and students of the robotic kits in education led to the release of even more robot kits [7]. In the commercial domain, the dominant robotic kit in education is the LEGO Education with the Mindstorm and Wedo series, followed by the Engino Robotics, Bee-Bot and Arduino-based kits. Some of these systems use a graphical representation of commands or low-level code [34,35].

As far as it concerns research works, authors in [36] introduced Hydra, a new low-cost educational framework with a custom printed circuit board (PCB) and modular electronic connections. Hydra is based on the Arduino micro-controller and also supports block-based coding through ArduBlockly. Its list of sensors is much smaller than that of FOSSBot. However, the authors of Hydra introduce a methodology for monitoring the progress of the educational process using the Petri Nets model, which can be very useful in future educational scenarios with robots.

FOSSBot has its origins in another open source and open design robot, which we designed in 2019 as part of the Google Summer of Code project. That robot, called Proteas (https://github.com/eellak/gsoc2019-diyrobot (accessed on 10 August 2022)), was a do-it-yourself robot, with 3D-printable parts and a modular design.

Although all of the previous solutions are suitable for specific ages or groups, students and their teachers often seek for a solution that can cover multiple learning needs and can be adopted by different ages and programming skill levels. Most of the commercial solutions do not offer that capability and limit the user to a specific range of capabilities. This limitation, in most cases, is part of the marketing strategy of the companies, which offer different version of their products to different age groups, each with its own, but limited capabilities. On the opposite side, FOSSBot is suitable for all ages due to its ability to boot different operation modes that cover the needs of different age groups, from kindergarten children to students. The closed structure of FOSSBot (shown in Figure 1), which protects all electronic devices and mechanical parts inside and does not leave anything exposed, makes it very safe for children of ages younger than 4 years old. However, the minimum child age that we target is 4 years, and the use of FOSSBot must always be under the supervision of an adult (parent or educator).

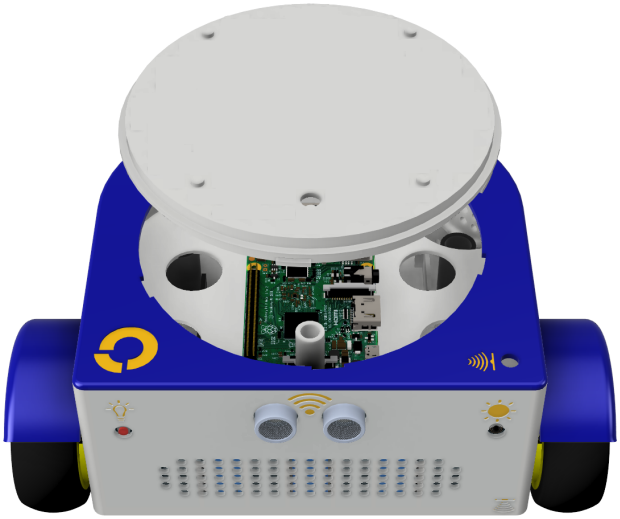

Figure 1. A front top view of the FOSSBot.

An open source and open design robot with low-cost electronics and printable parts that supports popular coding environments can be the basis for designing and delivering a wide range of educational solutions. It can support various STEM activities and help students and their educators to play, experiment and learn.

3. Proposed System

From the hardware point of view, FOSSBot is a complete end-to-end robot that employs easy-to-find, low-cost commercial electronics, and easy-to-print-and-assemble plastic parts. It has been designed to work on Raspberry Pi Zero, which provides internet connectivity, supports a wide range of sensors and allows the running of complex operations, such as robot control and computer vision tasks, on its micro-processor. Running on Raspberry Pi's operating system, the software stack of FOSSBot was developed using the latest trends

in software development, such as micro-services logic, dockerization and continuous integration (CI).

3.1. 3D-Printable Parts

Every plastic part of the robot, apart from the wheels, is 3D-printable and was designed from scratch in such a way that every sensitive electronic component or sensor is housed inside the robot in special sockets or specially designed containers. For example, a special container was designed for each wheel motor, which allows it to be quickly and steadily mounted to the main body of the robot. The same container has a special position to mount the odometer. The total time needed to print all of the plastic robot parts in a commercial 3D printer (e.g., Wanhao Duplicator 9) is approximately 36 h.

We employed PETG filament for our prints. PETG stands for polyethylene terephthalate glycol and is resistant to heat, impact and solvents. However, any other #D print filament alternative (ABS, ASA or PLA) can be employed, given that it provides strength to the printed parts.

The main body is the biggest part of the robot and was designed to ease the assembly of the different parts. In its interior there are predefined spaces for every electronic part. The external surface has printed symbols that indicate the position of every sensor. The symbols can be very useful to educators, since they provide an easy way to track every sensor and other electronic parts (e.g., speaker, charging plot, etc.). The front and upper side of the robot have printed grids that assist the cooling of the electronic parts. The charging port is located in the back side of the robot, along with the On/Off switch and a special loop that allows the towing of small items. Another purpose of this loop is to protect the robot from minor crashes. The second purpose is for courses that demand the robot to pull small objects. The main body also contains a vertical tube that runs from top to bottom and allows a pencil or marker to be attached to the robot. This allows the drawing of shapes in the ground by moving FOSSBot over a paper-covered area. Two spoilers were printed to the left and right of the robot in order to protect the wheels from side crashes and to add to the aesthetic design of FOSSBot.

The top surface of the robot is made up of two parts. The first part is a cover with special clips that is attached to the main body. The cover has a big circular cut in the middle with small rectangular cuts, which allow the main cover to be attached, using a place-and-rotate move, and lock. This main cover is easily detachable in order to provide access to the interior of the robot and, at the same time, can bear a LEGO bricks basis, allowing for more bricks to be attached on top of the robot. This option allows educators in the lower grades to use FOSSBot along with other LEGO activities, and can help in extending FOSSBot with more detachable electronic components that will be designed and developed in the future.

3.2. Electronics

The choice of suitable electronic parts for FOSSBot is not an easy task. It is important to use electronics that are common in the market, low-cost and compatible with Raspberry Pi. The list of electronics, as shown in Figure 2, comprises the following parts: gyroscope, accelerometer, two odometers, two motors for the wheels (the third standing point is a ball caster), RGB LED, rechargeable battery system, photo resistor, battery sensor to measure the power of the rechargeable batteries, ultrasonic distance sensor, speaker and, finally, gap/distance infrared sensor. The use of low cost and easy to find electronics gives the opportunity to everyone to buy and assemble FOSSBot with less than USD 200, and also maximizes the ability to repair the robot by replacing the defective part or by printing the worn plastic part. The different sensors and actuators can give the educators the ability to create numerous courses and activities that cover a wide range of disciplines.

One of the biggest challenges was to choose a suitable processor for FOSSBot that can support all sensors and provide an expandable platform for hosting our software stack and developing various educational activities [37]. Our programming language was Python

and the most promising platform choices were Raspberry Pi (Zero or 4), Arduino MKR and ESP32, which support Wi-Fi connectivity and are frequently used by robot software developers. Although Arduino MKR and ESP32 are frequently used in automation and robotics educational projects, they use a mobile version of Python programming language, which limits the ability to easily develop a composite software stack. Choosing Raspberry Pi Zero allows for the use of the full version of Python programming language, combined with a series of stable and handy libraries. Finally, the operating system of Raspberry Pi (Raspbian) supports software tools, such as Docker, which are very useful in the continuous integration and updates of the software stack.

Figure 2. The suite of FOSSBot electronics.

4. The Software Stack of FOSSBot

FOSSBot is based on a modular software stack, which is illustrated in Figure 3, that allows for (i) the implementation of various operation/programming modes, (ii) the orchestration of everything through its administration GUI and (iii) the controlling of the hardware in an easy way through a software library that plays the role of the FOSSBot operating system.

Figure 3. The software stack.

The software stack of FOSSBot comprises the following interconnected components: Google Blockly (https://developers.google.com/blockly (accessed on 10 August 2022)), Python Jupyter (https://jupyter.org/ (accessed on 10 August 2022)), Python Flask (https://flask.palletsprojects.com/en/2.1.x/ (accessed on 10 August 2022)), which hosts the administration/supervisor GUI of FOSSBot, the core FOSSBot library written in Python that controls the robot hardware, and, finally, the manual operation mode UI, which provides a friendly interface for users to control the robot without any knowledge of programming.

The core code of FOSSBot is written in Python and allows us to control the main actions of the robot (i.e. movement, sensors, actuators, sanity checks). The core library of the robot uses a high level of abstraction and offers an easy way to command the electronic parts of the robot. The core library is accessible by all other operation/programming modules of the robot.

The administration/supervisor container is responsible for ensuring the normal operation of the robot and provides the end user a way to modify default parameters and maintain them from the software aspect of the system.

The operation/programming modules are further detailed in the following sections.

A major open source project and community that develops robotic software is the Robot Operating System (ROS (https://www.ros.org/ (accessed on 10 August 2022))). ROS also adopts a modular architecture that organizes code in packages for basic operations, movement, sensors and data analysis, etc. It also defines a middleware interface between ROS and any specific middleware implementation. The current list of FOSSBot's sensors and electronics mostly supports the movement of the robot in 2D and some basic sensor data collection activities. Respectively, the OS of FOSSBot mainly wraps up basic movement functions (move forward/backward and turn) and sensor interfaces for collecting data. More advanced packages, such as path planning, computer vision, etc., are still not supported by the FOSSBot OS. We are currently developing packages for logging and diagnostics in order to support testing and quality assurance. The core FOSSBot OS library is also called by a middleware interface in the case of the no-coding and block-based coding modes, which wraps the main functionalities in REST API calls. The management interface also interacts with the robot through the same middleware interface.

4.1. Operation Modes

Through the administration panel, the users can choose to use the robot in one of the three following modes: (i) the no-coding UI, which is suitable for pre-school children and mainly demonstrates the robot's abilities, (ii) the block-based coding UI, which targets primary school students, and (iii) the notebook coding mode, which can be used to teach high school students the basics of programming (e.g., loops, conditions, events, etc.) in Python. In addition to these three modes, it is possible for more experienced users to write Python scripts directly on the FOSSBot programming shell, thus achieving a low-level control of the robot and its electronics. We call this mode the Python programming mode.

4.1.1. No-Coding Mode

This mode is addressed to users with little or no coding experience. The robot can be operated using buttons that appear in the main robot control screen. The screen contains, by default, four buttons that can move the robot forward and backward and turn the robot clockwise and anti-clockwise. The length of a forward of backward step and the degrees of the rotation are defined through the administration panel, but the default is 10 cm and 90 degrees, respectively.

This mode also allows for the addition of more buttons with pre-defined or newly coded functionality, as shown in Figure 4. For example, the educator can add more buttons that implement steps of different length, play a sound or turn on the LED light of the robot. The buttons can also be connected with more complex scripts, coded using the block-based coding UI. For example, the educator can prepare a follow-the-line script in the block-based coding UI, save it and link it to a button using FOSSBot's administration panel.

The ability to expand the basic set of buttons with more specially crafted buttons can keep the interest of students high, with new scenarios and activities. It also allows for the complexity of the course and activities to be easily adjusted based on the students' level and learning pace.

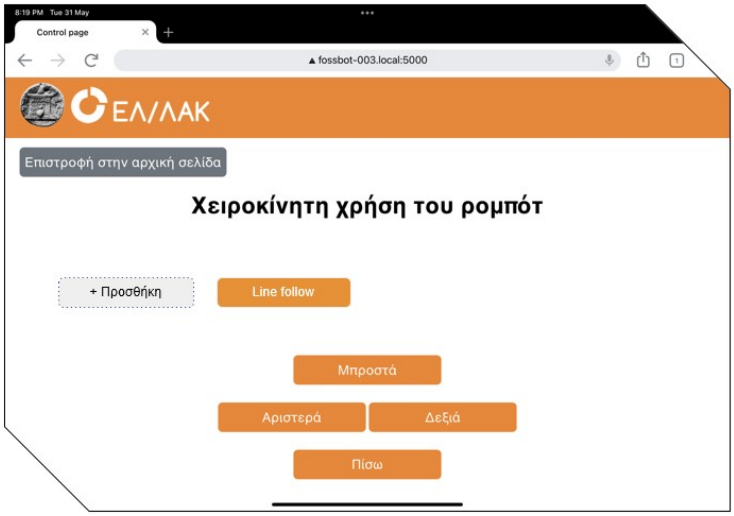

Figure 4. No-coding UI preview.

4.1.2. Block-Based Coding Mode

The block-based coding interface follows the successful paradigm of Scratch software (https://scratch.mit.edu/ (accessed on 10 August 2022)) in education. However, FOSSBot's UI is based on the open source Google Blockly software.

In the block-based coding mode, the user can create full operational programs using coding blocks, as shown in Figure 5. Every block corresponds to a command, which can also be parametrized, take one or more inputs and provide output and have a special role in the program, and can be connected or embedded in other blocks like pieces of a puzzle. The proper combination and arrangement of coding blocks builds the coding script.

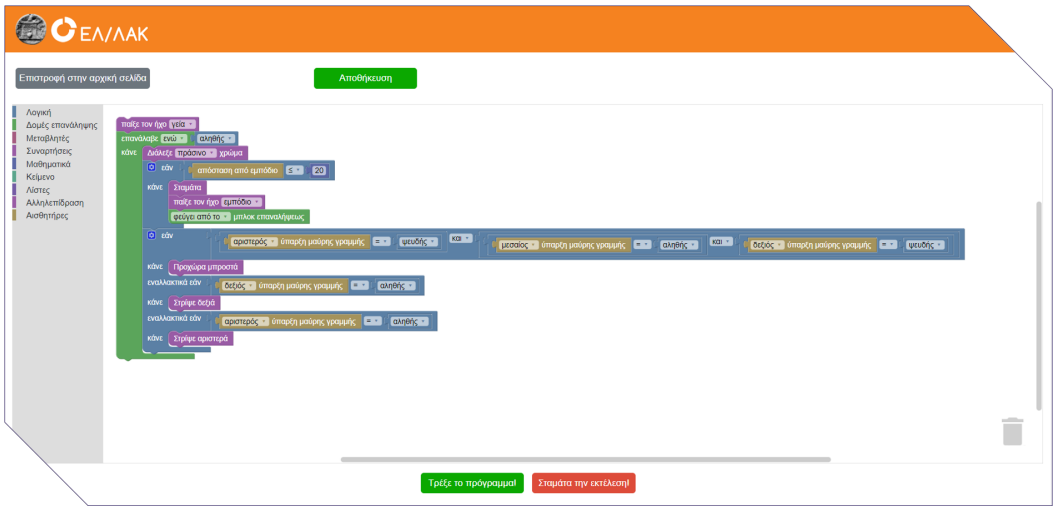

Figure 5. Block-based coding UI preview.

All of the block-based scripts can be stored in the robot's memory storage (i.e., an SD memory card) and retrieved at any time. They can be opened, modified and executed. They can also be linked to buttons in the no-coding mode. When the user decides to create a new script, a new screen with the Blockly UI appears that contains an empty script-construction area on the right and a sidebar on the left that groups all of the available commands in categories (e.g., loops, conditions, functions, etc.). The block categories comprise buttons commonly used in coding and mathematics.

What is unique here is a group of Blockly blocks that map directly to FOSSBot commands. These blocks are programmed in Python and call methods of the core FOSSBot library that control the sensors or the movement of the robot. More specifically, blocks in

the Movement category control the movement and rotation of the robot (e.g., keep moving forever, or move for a certain distance) and blocks in the Sensor category allow for the interaction with the sensors (e.g., to measure the distance from an obstacle). Finally, a third category of Interaction blocks controls how FOSSBot interacts and communicates with the user by blinking or changing the color of the RGB LED or by recording and playing an audio message from the speaker.

Developing a script using the block-based UI is a straightforward process of connecting the building blocks. The execution of the script is performed directly on the robot. Behind the scenes, the Blockly compiler module converts the block-based script to an executable Python script, which is executed in FOSSBot's backend.

4.1.3. Notebook Programming Mode

In this mode, FOSSBot can be transformed from an educational toy robot to an advanced educational tool that can support various activities, with emphasis on Python programming. The robot can be programmed using the Python programming language, which is combined with the high-level abstraction of the robot, as defined in the core FOSSBot library. Using the notebook mode, a Jupyter notebook opens that allows for the combining of Python commands that are executed directly on FOSSBot, with their output and with notes that describe and explain the whole process. Additional data management and visualisation commands provided by the popular Python libraries of Pandas and Matplotlib allow it to collect data from the sensors, analyze them and create a visual plot for the users. The final result looks like what is depicted in Figure 6, where an introductory paragraph about the human brain reaction time is followed by a script that runs on FOSSBot, prompts the way the user responds to a visual signal and records his/her reaction time.

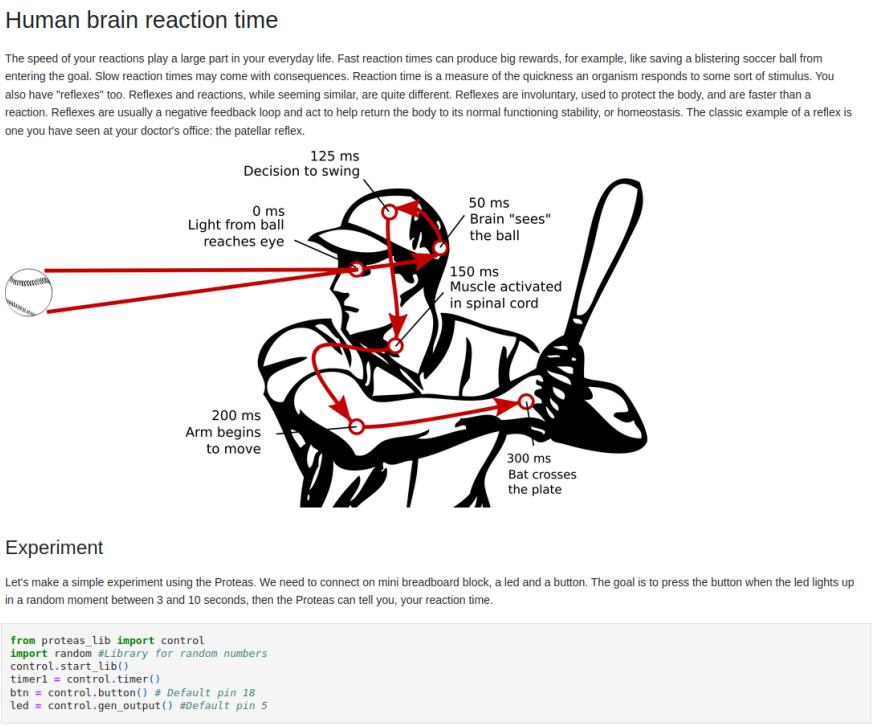

Figure 6. Notebook coding mode example.

The Jupyter notebook is a tool that is commonly used by data scientist and educators because of it browser-based UI and its ability to execute Python commands and directly display their output. FOSSBot's notebook IDE runs directly on the robot and can be accessed by any device (i.e., laptop or PC) that is connected in the same network as FOSSBot. The interactive nature of the notebook coding mode can be very appealing for students, who

have an unlimited number of ways to use FOSSBot's sensors, capture and analyze their data and set up their own experiments.

4.1.4. Python Programming Mode

The fourth and more advanced operation mode is by directly accessing the operating system of the robot. This can be achieved by directly connecting to FOSSBot's shell using an SSH connection. Then, the user can launch the Python environment and start writing Python scripts that are executed directly on FOSSBot. This mode is suitable for experienced users or researchers who wish to extend the capabilities of the robot by adding to its core library, or who simply want to have direct access to all sensors and electronics. This is the same mode used by the development team of FOSSBot for developing new functionalities or for adding new options to the administration module, to the Blockly-based programming mode, etc. A screenshot of the code behind the core robot library is depicted in Figure 7.

```
class motor():
    '''
    The motor Class create new motor objects. For every motor you should create a new object
    e.x. left_motor = motor(pwm pin,pin a ,pin b) also you can insert the frequense as freq of pwm
    and duty cicle as dc. The dc parameter controls the speed of the motor and accepts values 0 - 100.
    By default the constructor set the frequence on 1000 HZ and the duty cicle on 50.
    '''

    def __init__(self,speed_pin,terma_pin,termb_pin,freq=10000,dc=50):
        GPIO.setup(speed_pin,GPIO.OUT)
        GPIO.setup(terma_pin,GPIO.OUT)
        GPIO.setup(termb_pin,GPIO.OUT)
        self.terma_pin = terma_pin
        self.termb_pin = termb_pin
        self.mot = GPIO.PWM(speed_pin,freq)
        self.freq = freq
        self.dc = dc
        self.dir_control("forward")
        self.mot.start(0)
    def control_speed(self,speed):
        self.mot.ChangeDutyCycle(speed)
    def set_speed(self,speed):
        self.dc = speed
    def dir_control(self,direction):
        if direction == "forward":
            GPIO.output(self.terma_pin,GPIO.HIGH)
            GPIO.output(self.termb_pin,GPIO.LOW)
        elif direction == "reverse":
            GPIO.output(self.terma_pin,GPIO.LOW)
            GPIO.output(self.termb_pin,GPIO.HIGH)
        else:
            print("Motor accepts only forward and reverse values")
    def move(self,direction ="forward"):
        self.dir_control(direction)
        self.mot.ChangeDutyCycle(self.dc)
    def stop(self):
        self.mot.ChangeDutyCycle(0)
```

Figure 7. Real coding mode example.

4.2. Continuous Software Development and Updates

FOSSBot adopts a modern system architecture that has each one of the stack components in a separate dockerized image, which allows for the fast deployment and continuous integration of software updates.

The whole architecture is based on the continuous integration (CI) concept and, for that reason, the GitHub code repository is employed for staging our code. GitHub is a great code repository with several code management and collaboration capabilities. In addition, it offers a special tool, the GitHub Actions, which offers the ability to the developers to trigger a sequence of actions after a successful commit and merge in their code. In the case of FOSSBot, these actions handle the update of the container files of the FOSSBot software for various processor architectures (ARMv7, ARMv6, ARM64), which are immediately published as new images in the Docker Hub (https://hub.docker.com/repository/docker/chronis10/fossbot_basic (accessed on 10 August 2022)). The latter is a publicly available repository of software images that allows everyone from everywhere to pull a software image and deploy it in his/her system without the need for further installation or configuration steps. This flow offers a great way to push updates to any FOSSBot around the world, and allows FOSSBot users to retrieve software updates at the click of a button. Figure 8 illustrates the aforementioned process.

The use of GitHub also allows the open source community developers to safely contribute to the robot's code and, at the same time, allows end-users to raise issues in the Open Issues section.

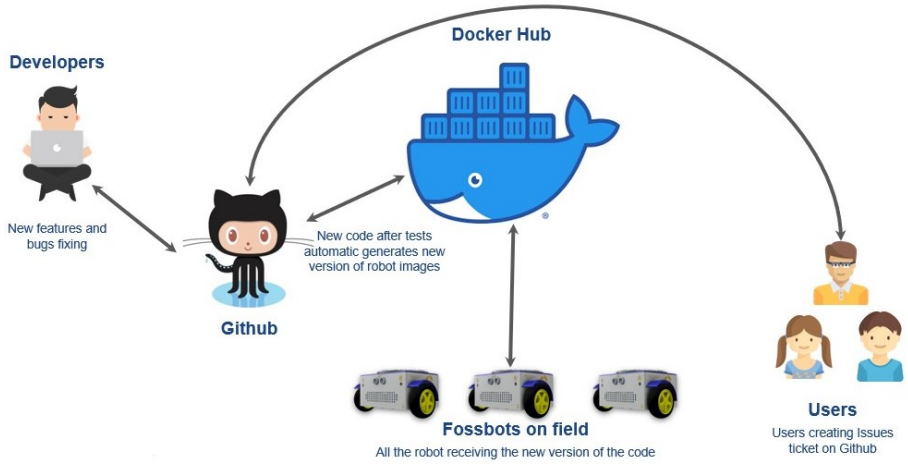

Figure 8. The software stack with the flow between robots, users and developers.

5. FOSSBot Pilot

As detailed in the previous sections, FOSSBot offers a multitude of ways to create educational activities for children and students of all ages. The different operation/programming modes and the variety of its sensors, electronics and other gadgets allow educators to use FOSSBot in various scenarios and teach various subjects, from programming and sciences to history or arts, through proper educational activities.

FOSSBot is currently supported by the Greek Free Open Source Software Society (GFOSS (https://eellak.ellak.gr/ (accessed on 10 August 2022))) and a huge effort has been made to assemble and distribute 100 robots to teachers in Greece, spanning all levels of education, in order to pilot FOSSBot. In close collaboration with a specialized team of STEM educators from GFOSS, the development team of FOSSBot carefully designs code blocks, extends the core library with functionality and continuously improves the administration UI in order to support the emerging educational needs and ideas. The experience of the educators with similar educational robots and kits provides valuable feedback for the design of the robot and new ideas for improving its integration into educational activities.

At the time of writing, the educators' team is working on the development of the FOSSBot usage manual and the educational material that will support courses for nursery schools and primary and secondary schools, as well as vocational training schools that will be trained on printing and assembling the robot. The nursery school material comprises courses and activities that use the no-coding mode and a special floor carpet with rectangles that teaches children how to navigate the robot, but also teaches them multiple topics by connecting the rectangles with domain specific concepts. Respectively, the primary school activities are simple, game-like activities designed on the Blockly mode, and so on. All of the educational material will be pre-loaded to the robot, and an additional external repository is under development, which will allow any educator that used FOSSBot to share his/her educational material with the community.

6. Conclusions and Next Steps

This work introduces FOSSBot, a new open source software and open design robot with a long list of sensors and several features that make it a powerful tool for educators who want to teach programming, sciences, arts and more topics at all educational levels.

In its first steps, FOSSBot has already found the support of open source communities and initiatives, such as the Greek FOSS society and the Onassis' foundation, which funded the assembly of the first 100 FOSSBots, the Google Summer of Code program, which funded

the first rounds of development, and, of course, the students and professors from Greek universities that embraced our effort. However, there is still space for improvement in order to make FOSSBot competitive to commercial solutions. Our main goal is to solve any issues related to the software of FOSSBot in order to simplify its assembly. In this direction, the next steps comprise the development of the educational material and the piloting of the robot, and the design of a board that will integrate most of the electronics' connections and will reduce the complexity of assembling the robot.

The development team is in close communication with educators, with the open source software and open hardware communities and FabLabs in Greece, in order to create a competitive and real useful tool for educators, students and robot enthusiasts. At the same time, we are currently working on porting FOSSBot to a robot simulation program (e.g., Gazebo or CoppeliaSim) so that anyone can use the same software stack and experiment with a virtual FOSSBot at no cost.

Last, but most importantly, we are currently working on optimizing all of the performance parameters of FOSSBot. As part of our next steps in this project, we plan an experimental evaluation in terms of computing efficiency, communication speed, control accuracy and other factors that require the careful design of the tasks and challenges, and a comparison with state-of-the-art solutions from the market and research.

Author Contributions: Conceptualization, I.V. and C.C.; software, C.C.; validation, I.V. and C.C.; writing—original draft preparation, C.C.; writing—review and editing, I.V.; supervision, I.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All of the code, data and material can be found at https://github.com/eellak/fossbot (accessed on 10 August 2022).

Acknowledgments: The authors would like to thank the students from Harokopio University of Athens who supported the development and assembly of FOSSBot so far: Eleftheria Papageorgiou, Thanassis Apostolidis, Dimitris Charitos, Giorgos Kazazis. In addition, the students from National Technical University of Athens who contributed with code during GSOC: George Yiannakoulias, Danai Brilli. Last but not least, the people from GFOSS and especially Thodoros Karounos who believed in FOSSBot and support our effort from its conception.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Evripidou, S.; Georgiou, K.; Doitsidis, L.; Amanatiadis, A.A.; Zinonos, Z.; Chatzichristofis, S.A. Educational robotics: Platforms, competitions and expected learning outcomes. *IEEE Access* **2020**, *8*, 219534–219562. [CrossRef]
2. Papert, S.; Harel, I. Situating constructionism. *Constructionism* **1991**, *36*, 1–11.
3. Benitti, F.B.V. Exploring the educational potential of robotics in schools: A systematic review. *Comput. Educ.* **2012**, *58*, 978–988. [CrossRef]
4. Kwon, D.Y.; Kim, H.S.; Shim, J.K.; Lee, W.G. Algorithmic bricks: A tangible robot programming tool for elementary school students. *IEEE Trans. Educ.* **2012**, *55*, 474–479. [CrossRef]
5. Wan, Z.H.; Jiang, Y.; Zhan, Y. STEM education in early childhood: A review of empirical studies. *Early Educ. Dev.* **2021**, *32*, 940–962. [CrossRef]
6. Li, Y.; Wang, K.; Xiao, Y.; Froyd, J.E. Research and trends in STEM education: A systematic review of journal publications. *Int. J. Stem Educ.* **2020**, *7*, 1–16. [CrossRef]
7. Alimisis, D. Educational robotics: Open questions and new challenges. *Themes Sci. Technol. Educ.* **2013**, *6*, 63–71.
8. Shen, V.R.; Yang, C.Y.; Wang, Y.Y.; Lin, Y.H. Application of high-level fuzzy Petri nets to educational grading system. *Expert Syst. Appl.* **2012**, *39*, 12935–12946. [CrossRef]
9. Stager, G. A constructionist approach to teaching with robotics. In Proceedings of the Constructionism and Creativity Conference, Paris, France, 16–20 August 2010; pp. 16–20.
10. Miglino, O.; Lund, H.H.; Cardaci, M. Robotics as an educational tool. *J. Interact. Learn. Res.* **1999**, *10*, 25–47.
11. Eguchi, A.; Shen, J. Student learning experience through CoSpace educational robotics. In *Proceedings of the Society for Information Technology & Teacher Education International Conference*; Association for the Advancement of Computing in Education (AACE): Asheville, NC, USA, 2012; pp. 19–24.
12. De Vries, M.J. *Handbook of Technology Education*; Springer: Berlin/Heidelberg, Germany, 2018.

13. Atmatzidou, S.; Demetriadis, S.; Nika, P. How does the degree of guidance support students' metacognitive and problem solving skills in educational robotics? *J. Sci. Educ. Technol.* **2018**, *27*, 70–85. [CrossRef]
14. Chin, K.Y.; Hong, Z.W.; Chen, Y.L. Impact of using an educational robot-based learning system on students' motivation in elementary education. *IEEE Trans. Learn. Technol.* **2014**, *7*, 333–345. [CrossRef]
15. Tuomi, P.; Multisilta, J.; Saarikoski, P.; Suominen, J. Coding skills as a success factor for a society. *Educ. Inf. Technol.* **2018**, *23*, 419–434. [CrossRef]
16. Yoel, S.R.; Dori, Y.J. FIRST High-School Students and FIRST Graduates: STEM Exposure and Career Choices. *IEEE Trans. Educ.* **2021**, *65*, 167–176. [CrossRef]
17. Caceres, P.C.; Venero, R.P.; Cordova, F.C. Tangible programming mechatronic interface for basic induction in programming. In Proceedings of the IEEE 2018 IEEE Global Engineering Education Conference (EDUCON), Canary Islands, Spain, 18–20 April 2018; pp. 183–190.
18. Sapounidis, T.; Demetriadis, S.; Stamelos, I. Evaluating children performance with graphical and tangible robot programming tools. *Pers. Ubiquitous Comput.* **2015**, *19*, 225–237. [CrossRef]
19. Bers, M.U.; Flannery, L.; Kazakoff, E.R.; Sullivan, A. Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Comput. Educ.* **2014**, *72*, 145–157. [CrossRef]
20. McGill, M.M. Learning to program with personal robots: Influences on student motivation. *ACM Trans. Comput. Educ. (TOCE)* **2012**, *12*, 1–32. [CrossRef]
21. Evripidou, S.; Amanatiadis, A.; Christodoulou, K.; Chatzichristofis, S.A. Introducing algorithmic thinking and sequencing using tangible robots. *IEEE Trans. Learn. Technol.* **2021**, *14*, 93–105. [CrossRef]
22. Kaburlasos, V.G.; Dardani, C.; Dimitrova, M.; Amanatiadis, A. Multi-robot engagement in special education: A preliminary study in autism. In Proceedings of the 2018 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 12–15 January 2018; pp. 1–2.
23. Amanatiadis, A.; Kaburlasos, V.G.; Dardani, C.; Chatzichristofis, S.A.; Mitropoulos, A. Social robots in special education: Creating dynamic interactions for optimal experience. *IEEE Consum. Electron. Mag.* **2020**, *9*, 39–45. [CrossRef]
24. Amanatiadis, A.; Kaburlasos, V.G.; Dardani, C.; Chatzichristofis, S.A. Interactive social robots in special education. In Proceedings of the 2017 IEEE 7th International Conference on Consumer Electronics-Berlin (ICCE-Berlin), Berlin, Germany, 3–6 September 2017; pp. 126–129.
25. Shim, J.; Kwon, D.; Lee, W. The effects of a robot game environment on computer programming education for elementary school students. *IEEE Trans. Educ.* **2016**, *60*, 164–172. [CrossRef]
26. Garris, R.; Ahlers, R.; Driskell, J.E. Games, motivation, and learning: A research and practice model. In *Simulation in Aviation Training*; Routledge: London, UK, 2017; pp. 475–501.
27. Feurzeig, W.; Lukas, G. LOGO—A programming language for teaching mathematics. *Educ. Technol.* **1972**, *12*, 39–46.
28. Suzuki, H.; Kato, H. AlgoBlock: A tangible programming language, a tool for collaborative learning. In Proceedings of the 4th European Logo Conference, Partenkirchen, Germany, 13–17 September 1993; pp. 297–303.
29. McNerney, T.S. From turtles to Tangible Programming Bricks: Explorations in physical language design. *Pers. Ubiquitous Comput.* **2004**, *8*, 326–337. [CrossRef]
30. Gallardo, D.; Julia, C.F.; Jorda, S. TurTan: A tangible programming language for creative exploration. In Proceedings of the 2008 3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems, Amsterdam, The Netherlands, 2–3 October 2008; pp. 89–92.
31. Wyeth, P.; Wyeth, G.F. Electronic Blocks: Tangible Programming Elements for Preschoolers. In Proceedings of the Human-Computer Interaction INTERACT '01: IFIP TC13 International Conference on Human-Computer Interaction, Tokyo, Japan, 9–13 July 2001; pp. 496–503.
32. Horn, M.S.; Jacob, R.J. Designing tangible programming languages for classroom use. In Proceedings of the 1st International Conference on Tangible and Embedded Interaction, Baton Rouge, LA, USA, 15–17 February 2007; pp. 159–162.
33. Horn, M.S.; Jacob, R.J. Tangible programming in the classroom with tern. In Proceedings of the CHI'07 Extended Abstracts on Human Factors in Computing Systems, San Jose, CA, USA, 7–12 May 2007; pp. 1965–1970.
34. Mondada, F.; Bonani, M.; Riedo, F.; Briod, M.; Pereyre, L.; Rétornaz, P.; Magnenat, S. Bringing robotics to formal education: The thymio open-source hardware robot. *IEEE Robot. Autom. Mag.* **2017**, *24*, 77–85. [CrossRef]
35. De Cristoforis, P.; Pedre, S.; Nitsche, M.; Fischer, T.; Pessacg, F.; Di Pietro, C. A behavior-based approach for educational robotics activities. *IEEE Trans. Educ.* **2012**, *56*, 61–66. [CrossRef]
36. Tsalmpouris, G.; Tsinarakis, G.; Gertsakis, N.; Chatzichristofis, S.A.; Doitsidis, L. HYDRA: Introducing a Low-Cost Framework for STEM Education Using Open Tools. *Electronics* **2021**, *10*, 3056. [CrossRef]
37. Evripidou, S.; Doitsidis, L.; Tsinarakis, G.; Zinonos, Z.; Chatzichristofis, S.A. Selecting a Robotic Platform for Education. In Proceedings of the 2022 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 6–8 January 2022; pp. 1–6.