Wireless Remote Control of Low-Cost Smart Devices for No-Coders

Leopoldo Armesto[1] [a], Sara Blanc[2] [b], Antonio González[3] [c] and Antonio Sala[3] [d]

[1]*Instituto de Diseño y Fabricación, Universitat Politécnica de València, Spain*

[2] *Instituto U. de Tecnologías de la Información y Comunicaciones, Universitat Politècnica de València, Spain*

[3]*Instituto U. de Automática e Informática Industrial, Universitat Politécnica de València, Spain*

Keywords: Block Programming, Educational Robotics, Robot Programming, Internet of Things, Home Automation.

Abstract: This paper describes the development of a block programming tool, named Facilino, to remotely control multiple agents consisting of low-cost smart devices based on ESP32 and Arduino, such an intelligent house and a robot, using Bluetooth and WiFi (HTTP) in the context of educational applications. Facilino is based on Blockly, a library that has been adapted to create Arduino code from custom blocks. The new tool is combined with App Inventor to develop Apps with no-coders. The paper describes the preliminary results using this tool within research and development and academic activities.

1 INTRODUCTION

In the last decades, the education system is quickly becoming outdated as a consequence of the introduction of new technologies, particularly the Internet of Things. We aim to contribute to a more modern school system by properly stimulating students' imagination and creativity through the adequate technological resources. Indeed, STEM approach to education (Schejbal et al., 2022; Ruutmann, 2015) provides a suitable field to increase students' motivation, level of participation and engagement, communication, collaboration, critical thinking and creativity.

Block programming tools have been popularized in the recent years coping with the dilemma "Learn to code" vs. "Code to learn". In the first approach, coding is the aim and students must learn their fundamentals, however this will be only of interest to a small part of the population; while in the second approach, students code with the purpose of learning an added value knowledge and coding becomes a tool for a given mean.

People have different opinion about the benefits of using block programming tools (Cash, 2020; Hadlow, 2018; Guzdial, 2022), but clearly its use has been growing in the last decade among young learn-

ers with few experience in coding, and multiple tools have been popularized among makers, educators and even senior people. Among them, Scratch (Maloney et al., 2010; Ouahbi et al., 2015) and Blockly (Pasternak et al., 2017; Trower and Gray, 2015) have become the more popular options.

Block programming tools have been widely used in different applications such as programming robots (Trower and Gray, 2015); on smart devices using MQTT in Internet of Things applications (Adi and Kitagawa, 2019) and industrial robots (Winterer et al., 2020). In addition to this, block programming tools have shown the capability to focus on the important concepts of coding, abstracting complex routines into a single block. They also have the advantage of hardware abstraction, at least, when referred to the programming of micro-controllers, since the block can have the same aspect, but the generated code might be adapted to the specific hardware requirements. Projects (Schejbal et al., 2022; Kusmin et al., 2019) based on Arduino, Raspberry Pi, ESP8266 and ESP32 processors allow students to participate on STEM project aimed with different purposes, such as robotics, light control, wearables, eco-technology, including also the development of soft-skills as part of the learning process. Block programming tools allow them to program these devices via Bluetooth or WiFi connectivity without getting stuck at details which are not relevant in terms of a STEM project. For instance, aspects such as remote data access between two devices might require complex concepts (for a

[a] https://orcid.org/0000-0003-0979-4428

[b] https://orcid.org/0000-0001-6439-2902

[c] https://orcid.org/0000-0002-4669-8374

[d] https://orcid.org/0000-0002-5691-8772

young learner) such as syncronization, baudrate transmission, telegram structure, data formatting, etc...

Facilino is a block programming tool based on Blockly (Armesto, 2016). It has been widely used by no-coder learners for programming smart devices in the Internet of Things (Armesto, 2019) and low-cost Arduino robots (Armesto, 2017). Recently, we have published a new version of Facilino, which is purely based on frontend-backend architecture developed using a variety of programming languages including PHP, javascript and HTML.

This paper focuses on Facilino's features such as bluetooth and WiFi communication with low-cost electronics in order to develop custom smart devices integrated with custom Apps using App Inventor 2. We have developed specific Facilino blocks for programming ESP32 devices and also Arduino (ATMega 328p with a Bluetooth module). In particular, we show the use of these new blocks on a low-cost robot, known as bPED and a smart-house (developed in the context of EcoThings project, grant 2021-1-ES01-KA220-SCH-000034349).

This paper is organised as follows: Section 2 describes the main developments of the paper using Facilino; Section 3 describes an extension for App Inventor compatible with Facilino blocks; Section 4 describes the results applied to remotely control a low-cost Arduino robot via bluetooth and a smart-house as well as a survey conducted to Facilino users. Section 5 draws some conclusions..

2 BLOCK PROGRAMMING WITH Facilino: COMMUNICATION BLOCKS

We have created several blocks in Facilino aimed to help students abstracting some complex aspects of communication between devices. For instance, on ESP32 processors, Facilino can generate code for communicating (using several approaches) with external devices via Bluetooth or WiFi using its integrated antenna, while for Arduino (ATMEGA 328p), wireless communication must be done using a Bluetooth/WiFi module (i.e.: HC-06 or ESP-01) through digital pins using a software serial library. Blocks used on either case are practically similar, while the code they generate is different (hardware abstraction). In the following, we will discuss some simple approaches to complete our aim.

2.1 Classic Bluetooth

In particular, for Bluetooth communication between two peer devices (using SPP profile (Argenox, 2020)), we propose two simple approaches[1]:

- **Commands:** A single byte is transmitted from one device to another. The number of commands is obviously limited to 256, but avoids synchronization between devices (every single byte is the actual data being transmitted). The device receiving commands will simply *listen* incoming data and proceed accordingly with received command. There must be an *agreement* between the transmitter sending data and the receiver interpreting commands.

- **Telegram:** One device can request data to another device and thus the receiver can provide a response is data is requested back. In this case, we can implement a basic telegram structure where data length can be larger than one byte length. In particular, we have implemented the following telegram structure STX+CMD+LEN+DATA+END, where STX is a telegram starting code (in our case we use '@' symbol), 'CMD' is the command number, 'LEN' is the length of 'DATA' field and 'END' is a telegram ending code (in our case we use '*' symbol). The length of all this fields is 1 byte, but the 'DATA' field that has a variable length, depending on the command.

Following this approach, we can distinguish between two types of telegrams on transmissions between an App controlling a smart device:

1. **App Requests for Sensor Readings:** The transmission is initiated by the App requesting data with a telegram, i.e.: requesting the read of a digital pin is implemented as @+0+1+10+* (command field is 0 and data length field is 1 and data is 10, that is the pin number). Then, the device receiving the telegram decodes it and sends a telegram back to the App with the data, i.e.: @+1+2+10+1+* (command field in the response is incremented by 1, data length is 2 and data is contains the read pin number and the actual pin value). Then App can notify the user on the reception of the response telegram with an event containing received data.

2. **App Requests for Actuators:** The transmission is initiated by the App with a telegram con-

---

[1]Of course, more complex approaches can be implemented too, but keeping it simple has the advantage that can be easily understood by pupils aged between 12-18 years old.

taining data to set the actuator value, i.e.: to control a servo position we can send the telegram @+16+2+3+90+*, where in this case, the servo position command is 16, data length is 2, and data field contains the pin number (3 in this example) and the target position in degrees (90º in this case). No response telegram is needed in this case.

2.2 REST API

In this approach, the device implements a web server receiving HTTP requests implementing a basic REST API (Fielding, 2000). In order to exchange data with the device, a client sends HTTP requests to specific REST end-points and the server returns a JSON file if read data is requested. For instance, reading a digital pin value can be done by sending an HTTP request to the REST end-point "DigitalRead/pin", where 'pin' is the pin number; the server will respond with a JSON file containing a 'status' field, 'pin' field and 'value' field. On the other hand, setting the value for a digital pin will imply to send an HTTP request to the REST end-poin "DigitalWrite/pin/value", where 'pin' is the pin number and 'value' is either 1 o 0 (true or false) and no response back is expected.

2.3 Communication Blocks in Facilino

Following the previous ideas, we have implemented Facilino blocks to allow bluetooth and WiFi communication. Figure 1 shows the block used to receive bluetooth commands, where the user can add up-to 256 commands and needs to set the command value. This block instruction is usually included inside the main loop of the code.
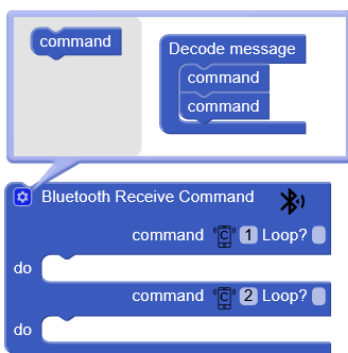

Figure 1: Facilino block for bluetooth communication via commands.

On the other hand, Figure 2 shows some of the blocks defined to decode telegrams on the device. The checkbox fields are used to enable decoding of

generic telegrams, such as digital or analog inputs or digital or analog outputs. More complex telegrams might require specific fields to access sensor data or to set the actuator value, such as the ones shown in the figure to read the temperature from a DHT sensor or to set the position of a servo.
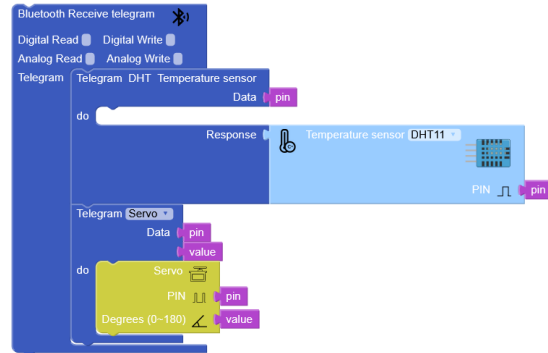

Figure 2: Example of bluetooth telegram decoding with Facilino. It decode telegrams to read a temperature as well as servo telegrams.

Also, Figure 3 shows some of the blocks defined to decode HTTP requrests on the device. As it can be seen, we have defined blocks that ressembles the bluetooth telegram blocks, but the generated code is completely different.
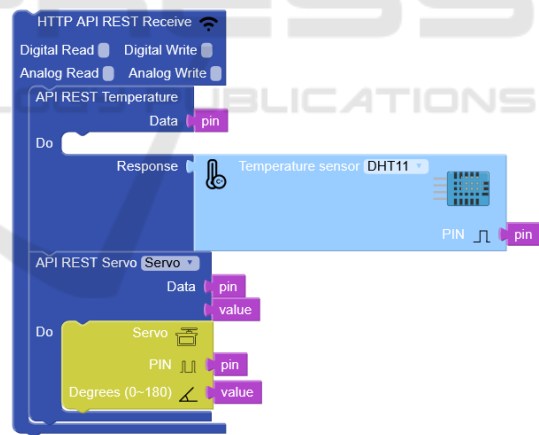

Figure 3: Example of REST API decoding with Facilino. It decode HTTP requests to read a temperature as well as servo requests.

In our current implementation, we have managed to implement telegrams for transmitting generic enumerated data such as booleans, integers, floats or strings. Also to control 180º and 360º servos, to set specific tones on a buzzer, to set a melody (a string concatenating frequencies and durations) on a buzzer, to set specific "expressions" on a 8x8 LED matrix or a 7-LEDs RGB strip and to read data from an ultrasonic sensor; and humidity and temperature from a

DHT sensor. Obviously, the number of telegrams can be extended to other sensors/actuators.

3 Facilino EXTENSION FOR APP INVENTOR

App Inventor 2 (AI2) is a web app developed by MIT App Inventor Team to develop Android Apps based on a simple set of instructions using Blockly, see, for instance to learn the fundamentals of AI2 (Wolber et al., 2011). In order to extend AI2 capabilities, they introduced the use of extensions (AI2, 2015), which allows developers to create new components that can be used within AI2. In that sense, we have created an AI2 extension to communicate with smart devices using Facilino (Armesto, 2015). In particular, two extensions have been developed in order to read from a set of predefined sensors or send command values to a set of actuators with use a very similar interface (see Figure 4 for the list of available components):

- **Bluetooth** extension: Based on the built-in Bluetooth client, it extends its functionalities implementing telegram coding/decoding to communicate with Facilino.

- **Web** extension: Based on the built-in Web client component, it extends its functionalities implementing a REST API communication with Facilino via HTTP.
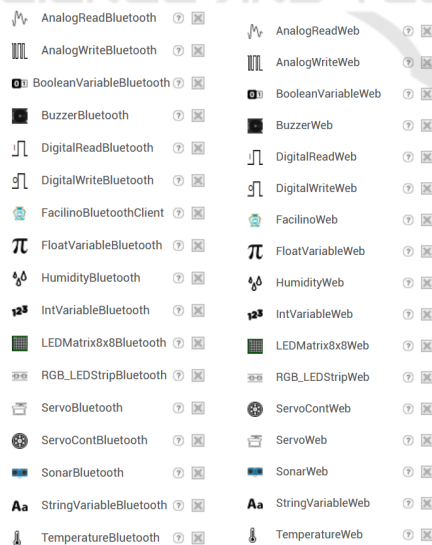
(a) Bluetooth extension   (b) Web extension

Figure 4: Components of Facilino AI2 extensions.

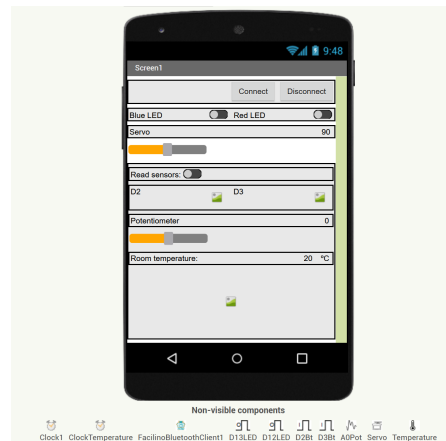Bluetooth extension uses *FacilinoBluetoothClient* component to handle bluetooth communication (con-

Figure 5: User-interface example on AI2 using Bluetooth extension.

nection and disconnection from the server), while the Web extension uses *FacilinoWeb* component to send/receive HTTP requests. In both cases, the extensions include equivalent components to send/receive generic data (such as booleans, ints, floats or string), to handle digital or analog signals or to exchange data with specific devices such as a humidity and temperature sensor, a sonar, a servo, RGB LED strip, etc...

All sensors implement an *Request* method to read data, using a non-blocking call. On data reception, the event *Received* notifies the user with the sensor value. On the other hand, actuators have procedures that allow to set their state, such as the *Set* method for a *DigitalWrite* component or the *Move* method for a *Servo* component. An example on how to use Facilino AI2 extension using bluetooth[2] can be seen in Figure 5 and Figure 6, where only relevant blocks have been included, while other blocks with replicated functionalities have been omitted for compactness. On the other hand, Figure 7 shows Facilino blocks used to read from digital and analog pins and a DHT sensor and to set LEDs values and position of a servo.

4 RESULTS

In order to validate our developments, we have tested Facilino communication capabilities on two test-bed educational platforms: a low-cost walking robot and a low-cost home automation prototype.

4.1 bPED: A Low-Cost Walking Robot

bPED (see Figure 8a) is a low-cost walking robot with 4 joints (two on each leg). It's movement is character-

---

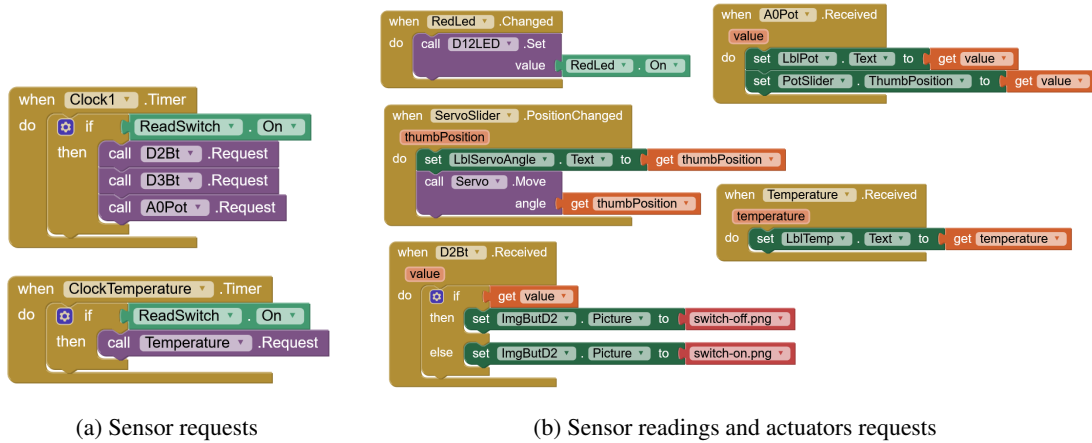[2]Web extension version uses equivalent components.

(a) Sensor requests   (b) Sensor readings and actuators requests

Figure 6: Main components for bluetooth App.
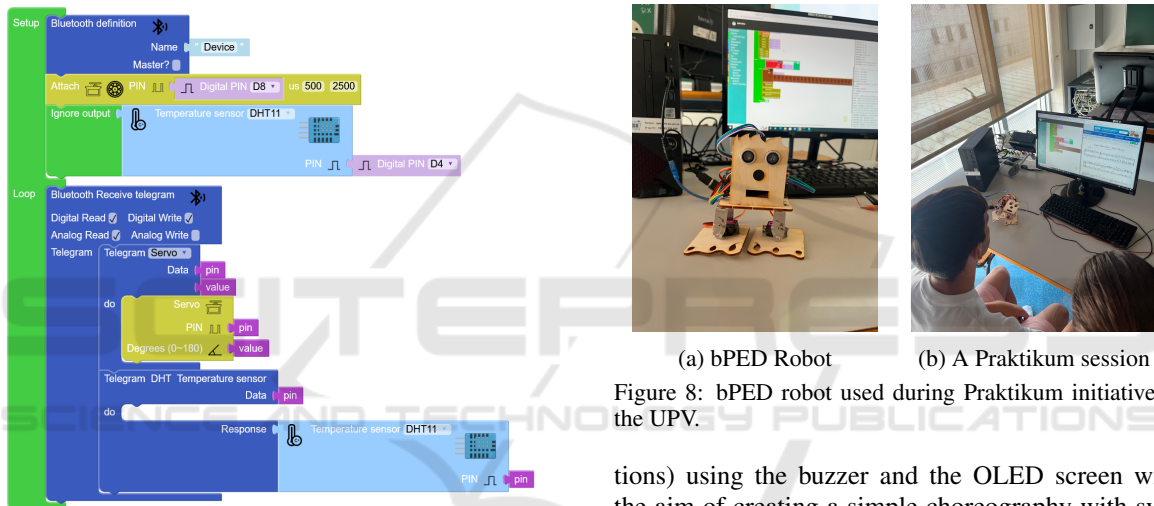
Figure 7: Facilino blocks using telegrams.

(a) bPED Robot   (b) A Praktikum session

Figure 8: bPED robot used during Praktikum initiative at the UPV.

ized by having no hip and thus in order to walk needs proper coordination of foot and leg motors on each leg. It uses an Arduino Nano (ATMega 328p) processor to control four servo motors (SG90) to move robot's joints, an ultrasonic sensor (HC SR04) to detect objects, a passive buzzer to produce sounds such as simple melodies, a 1.9' OLED screen to display images and a bluetooth module HC-06 to extend communication capabilities of Arduino Nano.

We have used this robot with a group of 8 students (17-years old) in a educational initiative, known as Praktikum, held at the Universitat Politècnica de Valencia (UPV) from 19th to 22nd june 2023. Within Praktikum, students received a 6h course on how to program with Facilino from scratch, using robot's components. They developed exercises to generate robot's movements, read from ultrasonic sensor, create customizable melodies and expressions (emo-
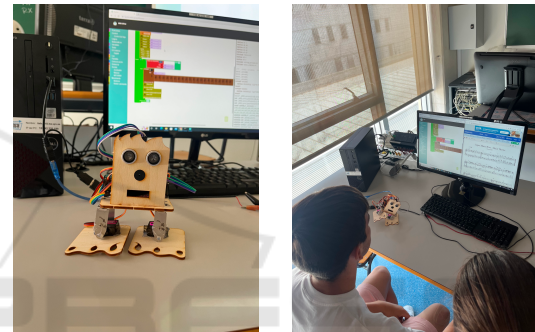
tions) using the buzzer and the OLED screen with the aim of creating a simple choreography with synchronized movements, sounds and emotions. In addition to this, they developed the code for a simple App containing buttons to send movement and expression commands to the robot. A sample App was provided to them with all components created for them, but with blank code (only with the user-interface), while the students had to implement code on AI2, but also the code on the Arduino side with Facilino (in Figure 10 code for one single command is shown for brevity, but students completed the code to include upto 17 movements and 13 expressions). To grasp an idea on the learning curve, students were able to complete all these activities, while, of course, enjoying creating their own programs. None of them had previous experience on Facilino, only three of them had previous experience with Arduino and also two of them used App Inventor beforehand in their regular academic activities.
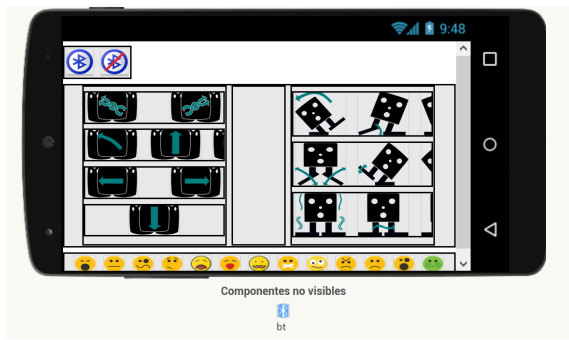
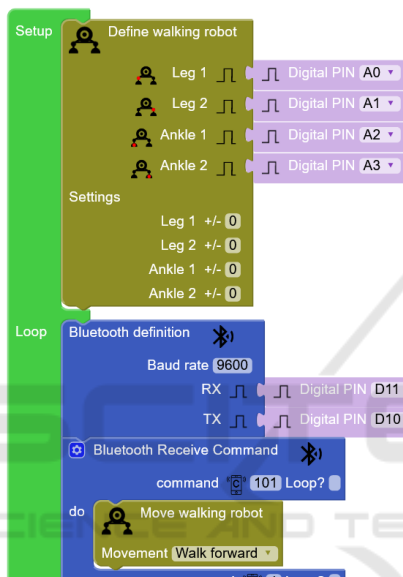Figure 9: AI2 user-interface with command buttons.

Figure 10: Facilino blocks for receiving bluetooth commands using bPED robot.

4.2 Home Automation

An intelligent eco-house described in this document has been designed under EcoThings Project. The purpose of this house is to serve as a mock-up in order to evaluate the set of proposed electronics in the context of an intelligent ecological house using similar principles of real houses and energy saving systems. As a consequence, the design has been oriented in providing low-cost solutions, considering as well as manufacturing aspects including 3D printing and laser cutting.

The smart house includes a living room and a green house that includes the following electronics: ESP32 board, Arduino Uno Extension Shield V5, 3 servo's (SG-90), Smoke/Gas sensor (MQ2), a mini fan, a temperature/humidity sensor (DHT11), a mini water pump moisture sensor, 7-RGB LEDs strip, LDR-sensor (KY-018), 2-Channel relay board and a

passive buzzer. Figure 11 shows the location of before mentioned electronics, while Figure 12 shows the implemented prototype manufactured using laser cutting with pywood and 3D printed parts. The prototype has been implemented by a group of Erasmus students during their stay at the UPV. In the video (Armesto, 2022), we show a working demo of the whole prototype.

We also have developed an App in App Inventor 2 integrating Facilino bluetooth extension. A couple of the screens developed for the App are shown in Figure 13, while part of the corresponding code for the screen Figure 13b is shown in Figure 14a, sending a number to indicate a predefined LEDs setting. Part of the corresponding Facilino code to decoding the telegram is shown in Figure 14b.

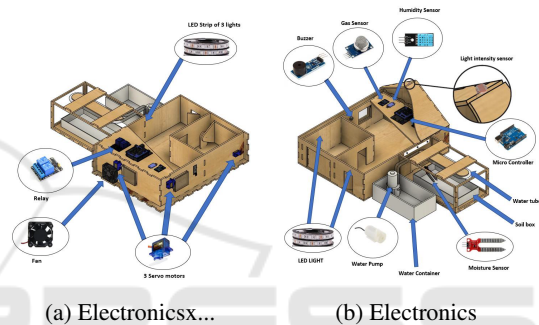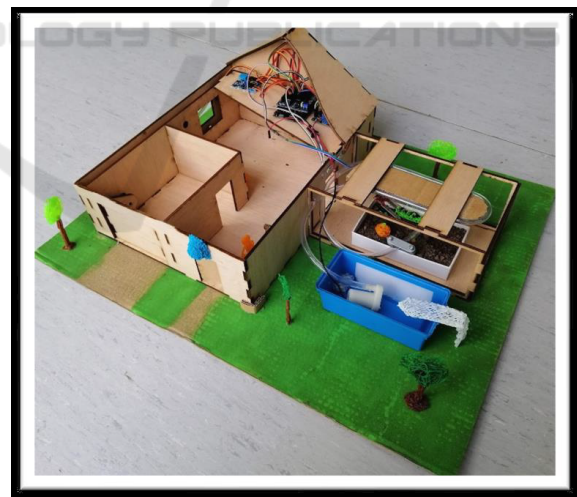(a) Electronicsx...   (b) Electronics

Figure 11: Electronics.

Figure 12: Eco-house prototype.

4.3 Survey

During this preliminary stage of the development, we have conducted a survey among 16 users that have tested the new version Facilino. In particular, the survey was filled by teachers from (latest courses of) pri-

(a) Window control screen
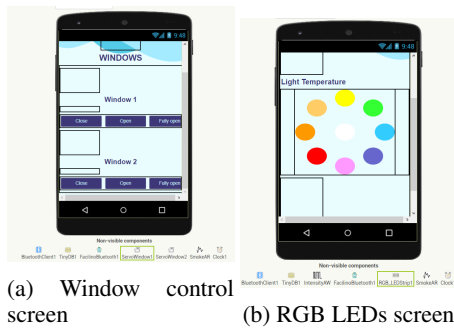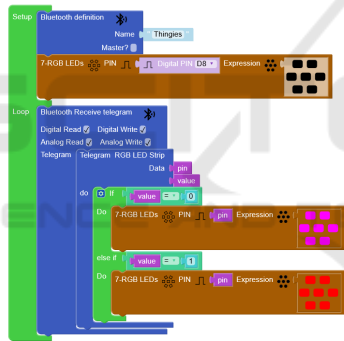
(b) RGB LEDs screen

Figure 13: Some of the screens developed in App Inventor 2 for the smart house project

(a) AI2 RGB LED code.

(b) Facilino RGB LED code

Figure 14: Example code for controlling RGB LEDs.

mary and secondary school levels. Only 9 of them have previous experience on programming with old Facilino version, but all of them have some experience and interest on block programming tools, that's why all of them considered very relevant the use of block programming tools appropriate to secondary level when they were asked for (this was a explicit question within the survey).

After testing the tool, they were asked about what kind of features considered more relevant and graded the main features of Facilino. In particular, they were asked about the following Facilino features: 1) Web application (front-end & back-end) that can be used from any device; 2) User accounts (capability to configure certain default aspects such as language and keep track of projects); 3) Over-the-Air (allow
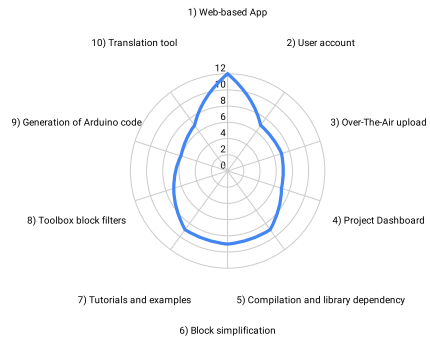

Figure 15: Relevance of Facilino's features.

to work from devices such as tablets or iPADs and upload code remotely to devices); 4) Project dashboard (allow to better organize the different types of projects and exercises requested in class, duplicate, share, etc...); 5) Easier code compilation, because all library dependencies are already included; 6) Block simplification (e.g.: inclusion of "shadow" blocks with default value for inputs); 7) Generation of tutorials and examples with built-in TinkerCAD simulations; 8) Toolbox blocks are adapted according to the type of project to be generated (capability to filer out some blocks which are not relevant for a project type); 9) Ability to view/modify the generated Arduino code before compiling and uploading the code to the microcontroller; 10) Translation tool (blocks are translated into several languages). As it can be seen in Figure 15 the feature that considered more relevant is the fact that Facilino works as a web application (multi-platform). Users consider quite relevant the fact that all compilation and library dependency issues have been simplified as well as the inclusion of tutorials and examples.

On the other hand, users were also asked to grade main features of Facilino with 1 being a poor grade and 5 the highest possible grade. As it can be seen from Figure 16, block programming obtains best grades, followed by the fact that users can create their custom project associated with their acccount and the inclusion of tutorials and examples. According to users, installation and setup needs to be improved, being the feature with the lowest grades.

5 CONCLUSIONS

In this paper, we have presented an on-going project regarding with the development of a block programming tool named Facilino. In particular, we have presented the proposed set of blocks to communicate with Bluetooth and WiFi with smart devices. We have shown a couple of case studies were Facilino has been
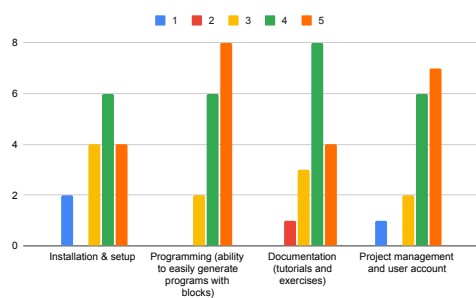
Figure 16: Grading of Facilino features (1-low, 5-high).

used in combination with an extension created for App Inventor 2 that allows users to remotely control smart devices based on Arduino or ESP32. The paper also shows a survey conducted to a small group of users. Although the number of users testing the tool is small, we took this survey to see potential flaws of the application for further improvement.

ACKNOWLEDGEMENTS

REFERENCES

Adi, P. D. P. and Kitagawa, A. (2019). A review of the blockly programming on m5stack board and mqtt based for programming education. In *2019 IEEE 11th International Conference on Engineering Education (ICEED)*, pages 102–107. IEEE.

AI2 (2015). Mit app inventor extensions. https://mit-cml. github.io/extensions/ [Accessed:05/07/2023].

Argenox (2020). Introduction to bluetooth classic. https: //shorturl.at/eLTY8 [Accessed:03/07/2023].

Armesto, L. (2015). Facilino extensions for ai2. https://github.com/roboticafacil/facilino_ai2 [Accessed:05/07/2023].

Armesto, L. (2016). Facilino. https://github.com/ roboticafacil/facilino [Accessed:29/06/2023].

Armesto, L. (2017). Diseña, fabrica y programa tu propio robot. https://www.edx.org/es/course/ disena-fabrica-y-programa-tu-propio-robot [Accessed:29/06/2023].

Armesto, L. (2019). Introduction to the internet of things. https://www.edx.org/course/ introduction-to-the-internet-of-things [Accessed:29/06/2023].

Armesto, L. (2022). Thingies. https://youtu.be/ 2fuLeuRaV4U [Accessed:29/06/2023].

Cash, J. (2020). Text-based vs. block-based coding. https://makelearn.org/2018/06/ 06/text-based-vs-block-based-coding/ [Accessed:29/06/2023].

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine.

Guzdial, M. (2022). Programming in blocks lets far more people code — but not like software engineers: Response to the ofsted report. http://bit.ly/3r83uau/ [Accessed:29/06/2023].

Hadlow, M. (2018). Visual programming - why it is a bad idea. http://mikehadlow.blogspot.com/2018/ 10/visual-programming-why-its-bad-idea.html [Accessed:29/06/2023.

Kusmin, M., Kusmin, K.-L., Laanpere, M., and Tomberg, V. (2019). *Engaging Students in Co-designing Wearable Enhanced Learning Kit for Schools*, pages 97–120. Springer.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., and Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–15.

Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., and Lahmine, S. (2015). Learning basic programming concepts by creating games with scratch programming environment. *Procedia-Social and Behavioral Sciences*, 191:1479–1482.

Pasternak, E., Fenichel, R., and Marshall, A. N. (2017). Tips for creating a block language with blockly. In *2017 IEEE blocks and beyond workshop (B&B)*, pages 21–24. IEEE.

Ruutmann, T. (2015). Optional stem courses for secondary schools designed and implemented for enhancement of k-12 technology education in order to excite students' interest in technology and engineering education. In *2014 International Conference on Interactive Collaborative Learning (ICL)*, pages 144–150.

Schejbal, A., Putyra, Å., Szemik, D., Zieliński, J., BasiuraD., Castilho, C., SilvaA., Costa, T., Rosa, C., SantosC., Pereira, H., Afonso, J., Olesk, P., Oja, M., and Park, V. (2022). Advancing stem education with iot experiments. https://est.edu.pl/iot/ wp-content/uploads/2022/12/IoT_Publication_EN.pdf [Accessed:03/07/2023].

Trower, J. and Gray, J. (2015). Blockly language creation and applications: Visual programming for media computation and bluetooth robotics control. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 5–5.

Winterer, M., Salomon, C., Köberle, J., Ramler, R., and Schittengruber, M. (2020). An expert review on the applicability of blockly for industrial robot programming. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1231–1234. IEEE.

Wolber, D., Abelson, H., Spertus, E., and Looney, L. (2011). *App inventor*. " O'Reilly Media, Inc.".