Received November 25, 2018, accepted January 23, 2019, date of publication January 29, 2019, date of current version February 14, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2895913

A Study on the Suitability of Visual Languages for Non-Expert Robot Programmers

JOSÉ MARÍA RODRÍGUEZ CORRAL[1], IVÁN RUÍZ-RUBE[1], ANTÓN CIVIT BALCELLS[2], JOSÉ MIGUEL MOTA-MACÍAS[1], ARTURO MORGADO-ESTÉVEZ[1], AND JUAN MANUEL DODERO[1]
[1]School of Engineering, University of Cádiz, 11519 Cádiz, Spain
[2]Technical School of Computer Engineering, University of Seville, 41012 Seville, Spain

Corresponding author: José María Rodríguez Corral (josemaria.rodriguez@uca.es)

This work was supported by the Spanish National Research Agency (AEI), under developed in the framework of the VISAIGLE Project with ERDF funds under Grant TIN2017-85797-R.

ABSTRACT A visual programming language allows users and developers to create programs by manipulating program elements graphically. Several studies have shown the benefits of visual languages for learning purposes and their applicability to robot programming. However, at present, there are not enough comparative studies on the suitability of textual and visual languages for this purpose. In this paper, we study if, as with a textual language, the use of a visual language could also be suitable in the context of robot programming and, if so, what the main advantages of using a visual language would be. For our experiments, we selected a sample of 60 individuals among students with adequate knowledge of procedural programming, that was divided into three groups. For the first group of 20 students, a learning scenario based on a textual object-oriented language was used for programming a specific commercial robotic ball with sensing, wireless communication, and output capabilities, whereas for the second and the third group, two learning scenarios based on visual languages were used for programming the robot. After taking a course for programming the robot in the corresponding learning scenario, each group was evaluated by completing three programming exercises related to the robot features (i.e. motion, lighting, and collision detection). Our results show that the students that worked with visual languages perceived a higher clarity level in their understanding of the course exposition, and a higher enjoyment level in the use of the programming environment. Moreover, they also achieved an overall better mark.

INDEX TERMS Authoring systems, educational robots, engineering education, interdisciplinary projects, visual programming.

I. INTRODUCTION

In order to make the creation of new applications easier for people without programming skills, recent software developments include programming tools that hide much of the complexity of traditional languages. On the other hand, it has been noted that students frequently experiment difficulties when learning traditional programming languages [1], [2] - such as C, C++ and Java - and abstract programming concepts [3].

A Visual Programming Language (VPL) [4] is any programming language that lets users create programs by manipulating program elements (variables, conditional statements, loops, functions, …) graphically rather than by defining them textually. They are systems where icons, symbols, charts and forms are used to specify a program [5]. Also, authoring tools allow a person to develop an application simply by linking together objects, such as a picture, a sound or a text paragraph. Authors can create useful and attractive graphics applications by simply describing the object's relationship among themselves and by sequencing them in an appropriate order [6].

In this sense, we can mention two tools: Scratch [7]–[9] for developing interactive stories, games, and animations, and App Inventor [10]–[12] for developing Android mobile applications. Both use a visual language based on interlocking visual elements which have to be assembled for specifying the behavior of the new applications. Those tools whereby programs are created by assembling visual blocks like the

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

pieces of a puzzle would be suitable for novice programmers [13], including those students in the first years whose university degrees are not directly related to the computer science discipline, such as sciences or industrial engineering.

VEDILS [14] stands for Visual Environment for Designing Interactive Learning Scenarios. It extends App Inventor allowing users to develop interactive, contextual and ubiquitous learning scenarios based on virtual and augmented reality technologies, as well as human-computer interface devices [15]. Recently, learning analytics techniques have been added to VEDILS [16], which provide the opportunity of developing applications for learning with monitoring capacities.

Currently, the use of robots for educative purposes is widely extended: Robotics is a multidisciplinary study field [17] which provides a learning environment that allows students to consolidate their knowledge in Mathematics, Physics, Electronic Engineering and Computer Science, among other disciplines [18], [19]. Also, there are a variety of development environments based on textual and visual languages for programming educational robots [20].

In particular, Sphero [21], [22] is a very attractive small robot which, in addition to its mobile capabilities, has internal lighting and also incorporates sensors for detecting collisions and orientation changes. It can be controlled using a computer or a smartphone, and its programming capabilities make this device a suitable tool for teaching programming languages [23].

Event-based programming [24] is the prevailing paradigm used in GUIs and other applications (e.g. JavaScript web applications), which perform specific actions in response to user input. In this paradigm, the program flow is driven by events such as user actions (key presses and mouse clicks), messages from other threads or programs, or sensor outputs. In this vein, a collision detection by a Sphero robot or the press of a button in a VEDILS application raises events that can be detected and handled by programs written in textual/visual programming languages with event-based programming features.

In view of the above, it is possible to think that VPLs make program understanding easier and program developing more enjoyable and intuitive because of their use of visual information. However, it is necessary to perform the corresponding experiments and obtain empirical and quantitative results that confirm such assumptions. In fact, Navarro-Prieto and Cañas [5] conduct an experiment to explain why and under what conditions VPLs would be easier to understand than textual programming languages. In such experiment, the mental models of spreadsheet and C programmers are evaluated in diverse program understanding situations, in order to conclude if VPLs enable a faster construction of a mental representation based on data flow relationships of a program than imperative languages.

An empirical study was carried out in [25] about the use by adult end-user programmers of a visual programming environment (*Modkit Alpha Editor*) for the Arduino open source hardware-software platform in comparison with its default textual environment. The authors concluded that visual environments help to create and modify programs, and provide a more positive user experience along with the perception of a reduced workload and a higher success.

In another work [26], Weintrop and Wilensky study high school students' perceptions about the blocks-based programming approach. Students in three sections of an introductory programming course took part in the experimental tests for the first ten weeks of the school year. The three classes participating in the study used different modified versions of the Snap! programming environment during the first five weeks of the course, and then transitioned to Java for the next five weeks of the study.

Students reported that several factors, such as the natural language labels on the blocks, their shapes and colors, the drag-and-drop composition interaction and the ease of browsing the blocks library contribute to making blocks-based programming easy. Moreover, they also identified drawbacks to blocks-based programming compared to the conventional text-based approach, including challenges in authoring larger and more sophisticated programs, issues of expressive power and a perceived lack of authenticity (i.e. to what extent the programming environment and practices conform to non-educational and conventional programming contexts).

Also, in [27] an analysis is performed about the completion of a simple programming activity by two groups of novice students, in order to compare the block interface of a programming environment with its textual counterpart. As a result, the students that used the block interface were able to meet a greater number of the activity goals in less time. Likewise, in another study [28] comparing block-based and text-based programming environments, the authors stated that no difference was found between students with regard to confidence in programming ability. However, students working with a block-based environment showed greater learning gains and a higher level of interest in future computing courses.

In addition, another work [29] analyzes how blocks impact the learnability of programming in contrast to textual languages. After reviewing studies on the effectiveness of block languages, the authors discuss their key features and how they relate to learning. Block programming rely on *recognition instead of recall*, reduce the cognitive load by *chunking code into a smaller number of meaningful elements* and provide *constrained direct manipulation of structures*, that help users assemble code without basic errors.

Moreover, various studies show the applicability of VPLs to robot programming [30]–[32]. However, at present, there are not enough comparative studies on the suitability of textual languages and VPLs for robot programming, although some research has already been conducted on this [33]. On the other hand, the lack of quantitative research on robot uses in education has been criticized in the past [34], [35].

The educational approach proposed in this work aims to introduce the students to the fields of robotics and visual programming [4], [5] from an eminently practical point of view, and also to familiarize them with event-based programming [24], thus leading to a learning experience in an interdisciplinary context. More precisely, we will try to answer - using quantitative and qualitative results - the following question:

Could the use of a VPL also be suitable in the context of robot programming as a "traditional" textual language (imperative or object-oriented) is? And, if so, What would be the main advantages of using a VPL?

This study aims to answer the questions raised above. In particular, the study performed in [33] compares a block-based interface developed by the authors (*CoBlox*) for programming a one-armed industrial robot with two development environments. The first one is based on a textual programming language (*RAPID*), where the programs are created and edited using the *ABB's Flex Pendant*, whereas the second is a menu-based programming tool (*Universal Robots' Polyscope*), where the programs are represented as a series of nodes in a hierarchical tree and written used a tree-based dialog driven strategy.

In this study, the participants were adults with little or no prior robotics programming experience. For the three environments, the participants used a simulated robot that performed the programmed instructions. They were asked to program the robot in order to accomplish a set of typical tasks of real-world robots, all of them based on a "pick and place" routine.

The main results of this work show that participants using *CoBlox* successfully completed more tasks and more quickly compared to those who used the other two environments. Moreover, participants using *CoBlox* reported greater levels of satisfaction, ease-of-learning and ease-of-use.

For our study, we have used a physical mobile robot, so that the cost of the device could be considered a disadvantage of our approach. However, the Sphero educational robot has an affordable price. Moreover, the capability of this type of robot for moving around in its environment allows us to design programming exercises of different nature to those designed for a robotic manipulator. In this sense, we can consider that our study is complementary to a certain extent. Also, the collision detection feature of the Sphero robotic ball allows us to design event-based programming exercises.

After stating the learning principles (Section II) and the technological foundations (Section III) on which this work is based, we describe those aspects related to the experimental tests: participants, design and procedure (Section IV). We started by developing three courses for programming the Sphero robot in three learning scenarios: Object-oriented Programming (OOP) in C# using the API for Sphero designed in [23], visual programming using the Sphero Edu app [36] and visual programming using the VEDILS authoring tool.

Next, we designed three programming exercises to be resolved in every learning scenario: the first one deals with Sphero mobile capabilities, the second exercise makes use of the robot internal lighting, and the last uses its sensing capabilities to detect collisions and execute an event handler method in order to provide a response to the collision events.

Once the three student groups - corresponding to the three learning scenarios - which participated in the experiment completed the three programming exercises, we proceeded to discuss the obtained results (Sections V and VI) and draw the conclusions according to them.

II. LEARNING PRINCIPLES

In this section, we discuss two principles that underlie the learning process of the students of the three experimental groups: motivation and interdisciplinarity. First, the students' motivation in their learning process is an essential aspect to be considered:

"Robotics has an inherent appeal on both emotional and intellectual level that makes it attractive to a broad range of learners across multiple dimensions, such as age, gender or academic interest" [18].

Also, App Inventor have turned students from consumers to developers of apps [10]. They have found enjoyable and stimulating to create apps for their phones or tablets. Likewise, a workshop in design of mobile apps using App Inventor [11] have helped participants to realize that they can build their own apps in order to fit their needs.

Moreover, these students participated in a learning activity where they had to apply, using an integrated approach, knowledge and skills from these disciplines: Robotics, Object-oriented Programming, Visual Programming and Event-based Programming [24]. A statistical analysis performed in [19] shows the increase of the perception in the relationship among robotics, mobile computing and many other related disciplines - computer vision, electrical engineering, embedded systems, mathematics, mechanical engineering, physics, computer architecture, human-computer interaction and operating systems - of a group of students who took part in a Systems Programming course combining mobile computing and robotics. Thus, interdisciplinarity is also a relevant learning principle in the context of this work:

"New thinking and innovation often occurs in the intersection between existing competencies and knowledge, and in the encounter between persons with different professional backgrounds. Firms are therefore seeking out knowledge workers who possess the ability to think across disciplines and to work together with others on common goals and tasks" [37].

III. TECHNOLOGICAL FOUNDATIONS

This section introduces the technological foundations this work is based on: the Sphero robot, the *SpherOOP* C# API [23], the *Sphero Edu* Android app [36] and the *VEDILS* authoring tool [14]). Sphero [21], [22], [38] is a ball-shaped robot [39] developed by *Sphero* (formerly *Orbotix*) to be controlled by a smartphone, a tablet or a computer through a Bluetooth link [40], [41]. In addition to its internal lighting, it can roll at a speed of about two meters per second. Its internal technology provides precision control and automatic stabilization capabilities, with a low center of mass in order to improve drivability and power efficiency. The purchased article includes an inductive charger providing a *charge-and-go* capability.

Sphero can stream data from the gyroscope for measuring the angular velocity rate, the inertial measurement unit in order to determine the orientation, and the accelerometer for detecting shake gestures and collisions [42]. This information can be used, for example, for enabling a Sphero device to act as a controller of other smart devices with network-available APIs which allow them to be controlled remotely [43].

The *SpherOOP* API developed in [23] provides a derived class that inherits all the features of the original *SpheroNET* [44] objects, but it adds new functionalities relating to motion and collision detection. *SpheroNET* consists of a wrapper for the Sphero low level API [45], [46] and it is based on the transmission of commands to a Sphero robot over a Bluetooth link.

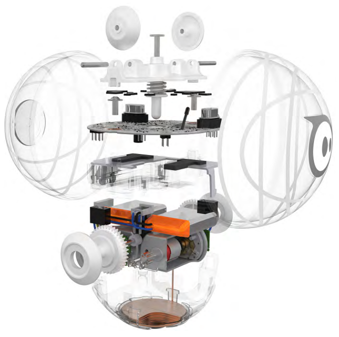


FIGURE 1. Inside Sphero SPRK edition.

Sphero SPRK Edition (Fig. 1) [38] is an educational version of the robot with the same electronics and sensors as Sphero 2.0. However, its transparent shell allows students to see all the internal mechanisms. The in-box accessories include a ‘‘genius kit’’ with pencil, notebook and protractor, instead of the ramps its predecessor came with. Also, Sphero SPRK+ Edition uses Bluetooth Smart technology, which makes the device connection process nearly instant, and comes with a strong scratch-resistant polycarbonate shell.

Sphero Edu [36] is the Sphero app for programming Sphero robots. Its block-based drag and drop interface for building apps allows students to learn the basic principles of programming in an easy and fun way (Fig. 2). This app also allows them to write JavaScript text programs that can directly control a Sphero device. Thus, once a program has been built as a block sequence, the equivalent JavaScript code can then be viewed.

App Inventor is a block-based visual programming tool developed by Google which allows everyone, even beginners, to start programming and create apps for Android devices. Currently, App Inventor is maintained by the Massachusetts Institute of Technology (MIT). Teachers and people in general can develop apps for running on smartphones and tablets using this authoring tool [10]. These apps can use the functionalities provided by Android devices: multimedia features (microphone, photo camera, video player,...), drawing and animation, sensors (accelerometer, gyroscope, GPS locator,...), web services, sharing data with social networks, etc.

The App Inventor tool structure consists of various elements: a Google Web Toolkit (GWT) application for designing the user interface of the apps, a block editor for programming their behavior by putting blocks together, a build server to pack the design and the logic into an Android Application Package (∗.apk), an interpreter that runs on the mobile device for debugging the apps and a module with all the built-in components (visuals and non-visuals), which are needed by the other modules and are available to the end users for developing their apps.

VEDILS project [14] (Fig. 3 and Fig. 4) has been built on top of App Inventor, extending it with modules that give teachers and educators the opportunity of including gestural interaction, augmented reality capabilities [15], robotics and learning analytics [16] in their mobile apps.

Fig. 3 and Fig. 4 show an example of use of the Leap Motion controller [47]. When the sensor detects a movement of any of the user’s fingers, the corresponding Android device sends a command to the Sphero robot in order to make it roll at a certain speed in the direction of the finger movement. If the user taps the Leap Motion controller with any of his or her fingers, the robot stops rolling.

Also, when the user puts his or her left hand (*hand 0*) over the Leap Motion controller, the robot vibrates/shakes ten times with a pause of four seconds between two consecutive vibrations. If the user takes his or her hand off the Leap Motion controller, the robot stops vibrating. Finally, when the user draws circles with any of his or her fingers over the Leap Motion controller, the Sphero device spins as many times as circles has been drawn by the user.

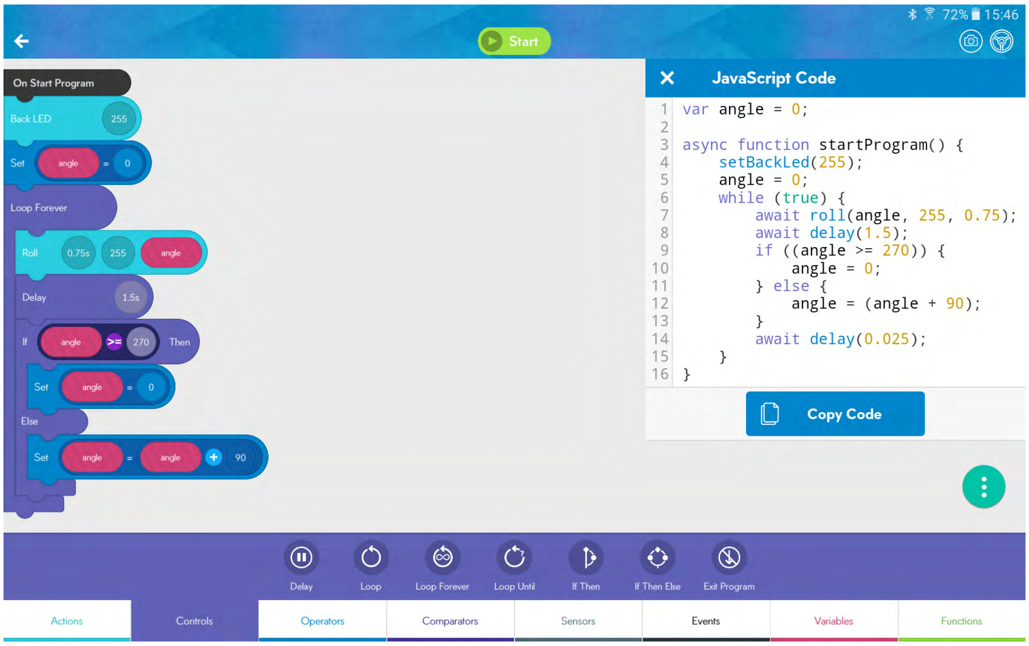
In order to integrate the control of the Sphero robot into the VEDILS tool, we have used the API [48] for Android devices developed by Sphero. A new non visible component (*SpheroControler*) has been developed, consisting of a Java wrapper class for the methods provided by the API (*ConvenienceRobot* [49]). Fig. 5 shows the class diagram corresponding to the implementation. It is important to note the use of the *ResponseListener* interface for handling the asynchronous messages sent by the Sphero device.

IV. EXPERIMENTAL TESTS

In order to provide answers to the questions that underlie our research, posed in Section I, we carried out a set of experimental tests about robot programming in three *learning scenarios* based on textual and visual programming. Finally, we made a comparative study of the results obtained with the purpose of drawing some conclusions.

A. PARTICIPANTS

The students of Fundamentals of Computer Science, imparted in the first year of the Degrees in Industrial Electronic, Industrial Technology, Electrical and Mechanical Engineering (University of Cadiz, School of Engineering), are



```
var angle = 0;

async function startProgram() {
    setBackLed(255);
    angle = 0;
    while (true) {
        await roll(angle, 255, 0.75);
        await delay(1.5);
        if ((angle >= 270)) {
            angle = 0;
        } else {
            angle = (angle + 90);
        }
        await delay(0.025);
    }
}
```

FIGURE 2. **"SquareRoll" program developed with *Sphero Edu* Android app.**

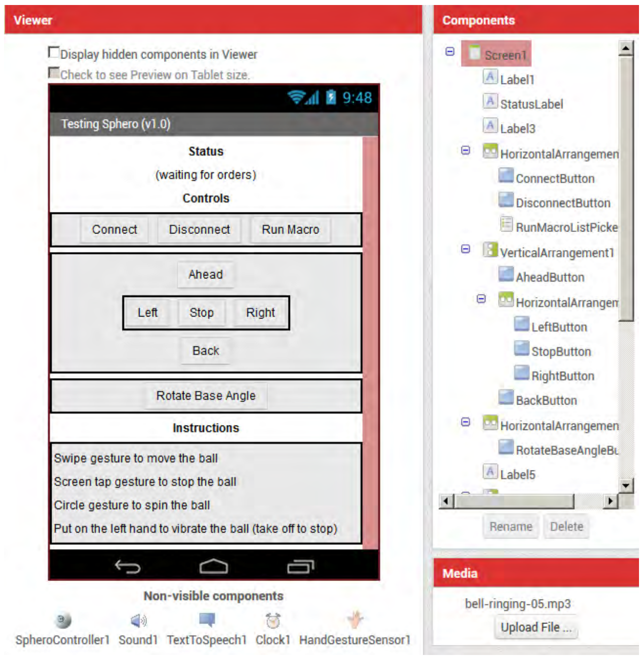


FIGURE 3. **Designer view.**

introduced to the C imperative programming language [50]. Thus, these students have been initiated in computer programming but they still do not have an in-depth knowledge about it. This starting condition makes them a suitable population for our research.

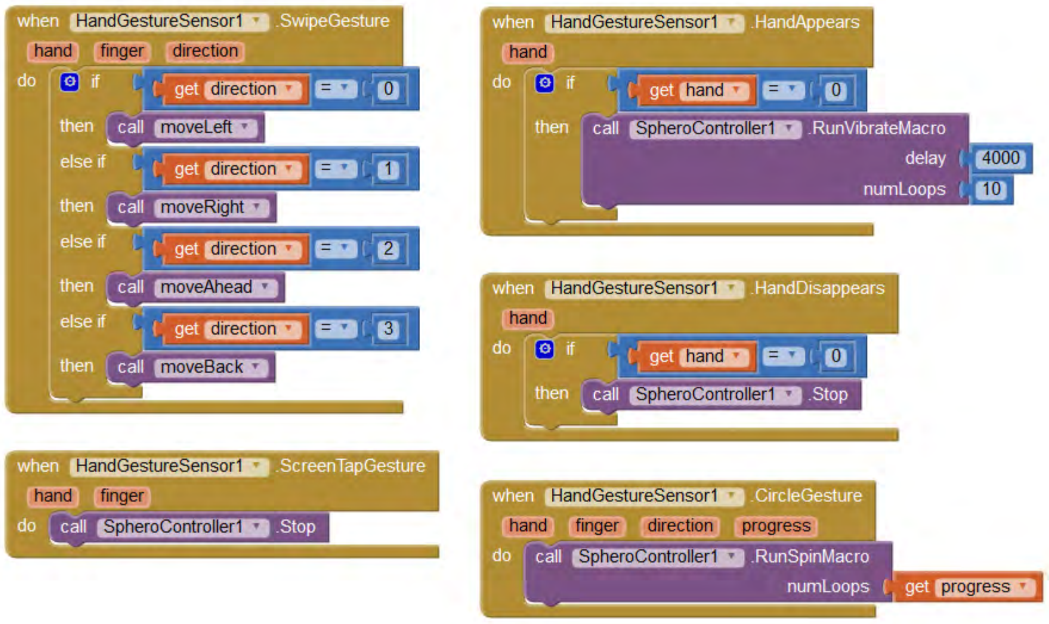
As a sample for our study, we selected those students who had a positive attitude toward learning and a high and similar performance level during the *Fundamentals of Computer Science* course. In this way, we can ensure that they were really interested in computer programming. We selected a sample of sixty students aged 18-19. All of them were men as the percentage of women studying Industrial Engineering (at least, in the University of Cadiz) is usually very low. None of these students had a previous contact with object-oriented programming, visual programming or robotics.

During the course and before the experiments were conducted, the students took an assessment test on C language programming. The selected students obtained an average mark of 8.2 points (out of 10) with a standard deviation of 0.7 points. These data show that the participants had a similar and relatively high programming skill level.

Considering that, in our case, the participants had already a basic knowledge of computer programming (use of constants and variables, conditional and iterative statements, etc.), the intended learning objectives were related to robot programming (the Sphero robotic ball in this case) and event-based programming.

Thus, at the end of the robot programming course the students should know and be able to use the necessary elements of the programming language used - textual or visual - in order to:

- Send angle and speed data to the robot so that it executes a specific trajectory (e.g. a circular trajectory).
- Send color data to the robot so that it changes the color of its internal RGB LED (e.g. to make the robot change successively its color every second among red, green and blue).
- Make the robot perform a specific task when its accelerometer detects a collision (e.g. to start rolling in the opposite direction to the original one).



```
when HandGestureSensor1 .SwipeGesture
  hand  finger  direction
do  if get direction = 0
    then call moveLeft
    else if get direction = 1
    then call moveRight
    else if get direction = 2
    then call moveAhead
    else if get direction = 3
    then call moveBack

when HandGestureSensor1 .HandAppears
  hand
do  if get hand = 0
    then call SpheroController1 .RunVibrateMacro
                                  delay 4000
                               numLoops 10

when HandGestureSensor1 .HandDisappears
  hand
do  if get hand = 0
    then call SpheroController1 .Stop

when HandGestureSensor1 .ScreenTapGesture
  hand  finger
do  call SpheroController1 .Stop

when HandGestureSensor1 .CircleGesture
  hand  finger  direction  progress
do  call SpheroController1 .RunSpinMacro
                        numLoops get progress
```

FIGURE 4. Block view.

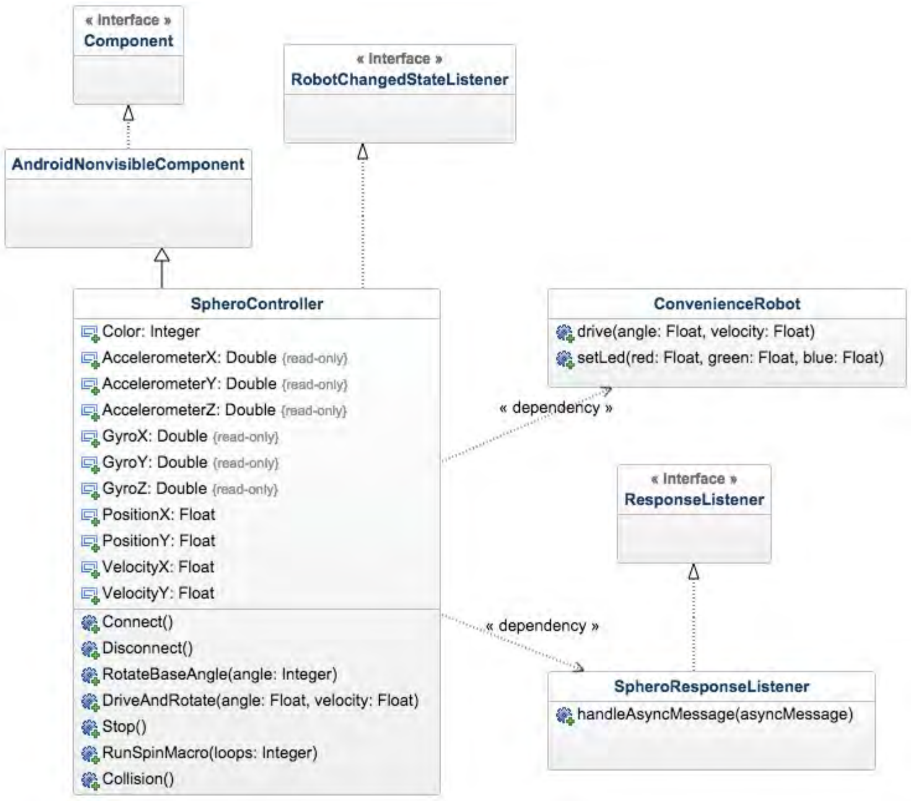

« Interface »
Component
« Interface »
RobotChangedStateListener
AndroidNonvisibleComponent
SpheroController
Color: Integer
AccelerometerX: Double {read-only}
AccelerometerY: Double {read-only}
AccelerometerZ: Double {read-only}
GyroX: Double {read-only}
GyroY: Double {read-only}
GyroZ: Double {read-only}
PositionX: Float
PositionY: Float
VelocityX: Float
VelocityY: Float
Connect()
Disconnect()
RotateBaseAngle(angle: Integer)
DriveAndRotate(angle: Float, velocity: Float)
Stop()
RunSpinMacro(loops: Integer)
Collision()
ConvenienceRobot
drive(angle: Float, velocity: Float)
setLed(red: Float, green: Float, blue: Float)
« dependency »
« Interface »
ResponseListener
« dependency »
SpheroResponseListener
handleAsyncMessage(asyncMessage)

FIGURE 5. Class diagram of the *SpheroController* VEDILS non visible component.

B. DESIGN

In this subsection, we will describe briefly the three courses for programming the Sphero robot referred in the introductory section of this article, which give raise to the three learning scenarios. Next, we will explain the programming exercises to be solved by the participants. Finally, we will define the variables used in the experimental tests. In order to carry out them, the total sample was divided into three groups of twenty students. A simple random criterion was used to distribute the sixty students among the groups. Each group attended a different course:

- Object-oriented programming: The students in the first group took part in a C# [51]–[53] OOP course using the *SpherOOP* API developed in [23] for controlling the Sphero robot through a PC.
- Visual programming using a mobile device: The students in the second group participated in a visual programming course using the *Sphero Edu* app for mobile devices [36] in order to control the robot from a tablet or an smartphone.
- Visual programming using an authoring tool: The students in the third group took part in a visual programming course using the VEDILS authoring tool for PCs in order to control the Sphero device through a tablet or an smartphone.

Each of the three groups of students had to solve, using the programming resources available in the specific learning scenario, the same three programming exercises: ‘‘SquareRoll’’, ‘‘RGBLed’’ and ‘‘MovingCollision’’. These exercises were related to the functionalities of the Sphero robot: motion, internal lighting and sensing capabilities.

The ‘‘SquareRoll’’ exercise uses the Sphero robot moving capabilities in order to describe a squared path. After the program (Fig. 2) has set Sphero back LED to its maximum intensity (255), the robot rolls at the maximum speed (255) during 0.75 seconds. Then, the robot stops and, after a delay of 1.5 seconds, the angle trajectory is incremented by 90 degrees and the robot starts rolling again. This cycle repeats indefinitely until the user stops the program execution. The direction of the trajectory is controlled by the ‘‘angle’’ variable, which successively takes the values 0, 90, 180 and 270.

The ‘‘RGBLed’’ exercise uses the Sphero internal lighting. After initializing the variables and the Bluetooth connection to the robot, the program changes its color to white for two seconds. Then, the main loop changes periodically the color of the robot - red, green and blue - and plays a different sound - ‘‘whistle’’, ‘‘ping’’ and ‘‘doorbell’’ - every two seconds. Fig. 6 shows the solution to the exercise written in C# using the *SpherOOP* API [23].

The ‘‘MovingCollision’’ exercise uses the Sphero capability of sending an asynchronous message when its accelerometer detects a collision. First, the program initializes the Bluetooth connection to the robot, as well as its color to red and the angle trajectory to zero degrees. Then, the program configures the Sphero device to raise an event when it detects an impact, and makes the robot roll at full speed.

The event handling method plays a sound and updates the robot color to the next one in a sequence composed by the colors red, green and blue, as well as the angle trajectory, that changes between 0 and 180 degrees. Finally, the robot is made roll in the opposite direction to the previous one. Fig. 7 shows the solution to the exercise implemented using VEDILS.

A ZIP package containing the material used for the experimental tests is available through the link https://goo.gl/7SJm6z. This material consists of the PDF presentations (in Spanish) for the three courses - a standard C# OOP course with example programs, an introduction to the use of the Sphero Edu app and a presentation about the VEDILS authoring tool -, the *SpherOOP* C# API source code for controlling the Sphero robot, the ‘‘MovingSphero.cs’’ demo code, developed using the Microsoft Visual Studio IDE, the ‘‘TestingSphero.aia’’ and ‘‘MovingSphero.aia’’ VEDILS demos, as well as the statements and solutions for the three exercises in the three learning scenarios. In addition, the sample data can be accessed through the IEEE DataPort repository using the DOI http://dx.doi.org/10.21227/5x1h-ad78. Finally, a short video-clip of the ‘‘MovingSphero.cs’’ C# demo execution can be watched accessing the link http://youtu.be/swPNGf8RqZI.

The quantitative results of the experimental tests were expressed as a set of indicators, calculated as the mean values of the data obtained for each learning scenario. The provided indicators for student perceptions were:

- Subjective perception of the clarity level (from 1 to 4) for the course exposition (*CL*).
- Subjective perception of the interest level (from 1 to 4) for the course exposition (*IT*).
- Subjective perception of the difficulty level (from 1 to 4) for exercises 1, 2 and 3 respectively (*DF1*, *DF2* and *DF3*).
- Subjective perception of the enjoyment level (from 1 to 4) for the development environment (*EJ*).

The provided indicators for student learning data were:

- Time spent (expressed in minutes) to study the course contents (*ST*).
- Achieved mark (from 0 to 2) for exercises 1, 2 and 3 respectively (*MR1*, *MR2* and *MR3*).
- Spent time (expressed in minutes) for exercises 1, 2 and 3 respectively (*TM1*, *TM2* and *TM3*).

C. PROCEDURE

Once the three courses for the three learning scenarios - C# OOP programming, Sphero Edu visual programming and VEDILS authoring tool - were taught, the three student groups were asked to indicate using a scale between one and four - to avoid the selection of neutral options - their perception of the clarity and the interest of the exposition (CL and IT indicators), as well as the time spent studying the course contents (ST indicator).

The instructor for the three groups was one of the authors of this work, and the total duration for the three courses was eight hours (four two-hour sessions).

All the students also undertook the same three programming exercises, which were marked using the logical sense of the algorithms and the implementation degree of the functionalities required by the statements as evaluation criteria,

```
using System;
using System.Drawing;
using System.Media;
using System.Threading;
using SpherOOP;

namespace RGBLed {
    class RGBLed {
        static void Main() {
            // Variable declarations
            Sphero sphero; // Sphero object
            // Set whistle sound
            SoundPlayer WhistleSound = new SoundPlayer("Whistle.wav");
            // Set ping sound
            SoundPlayer PingSound = new SoundPlayer("Ping.wav");
             // Set doorbell sound
            SoundPlayer DoorbellSound = new SoundPlayer("Doorbell.wav");
            int i = 0;

            // Connect to first available Sphero
            sphero = Sphero.Connect();
            if (sphero == null) { // Error connection
                Console.WriteLine("Can't connect to any Sphero device!");
                Console.WriteLine("Press any key to exit...");
                Console.ReadKey(true);
            }
            else {
                // Change color to white and switch on back LED
                sphero.SetRGBLEDOutput(Color.White.ToArgb());
                Thread.Sleep(2000); // Pause of two seconds

                // Change color to red, green and blue in turn
                while (true) {
                    if (i == 0) {
                        sphero.SetRGBLEDOutput(Color.Red.ToArgb());
                        WhistleSound.Play();
                    } else if (i == 1) {
                        sphero.SetRGBLEDOutput(Color.Green.ToArgb());
                        PingSound.Play();
                    } else if (i == 2) {
                        sphero.SetRGBLEDOutput(Color.Blue.ToArgb());
                        DoorbellSound.Play();
                    }
                    if (i >= 2) i = 0;
                    else i++;
                    Thread.Sleep(2000);
                }
            }
        }
    }
}
```

FIGURE 6. **"RGBLed" program developed with the *SpherOOP* C# API.**

and a scale between 0 and 2 (0 - Poor: The student did not resolve the exercise at all and the code has no logical sense, 1 - Average: The student only solved part of the exercise and part of the code has logical sense, 2 - Good: The student solved the whole exercise and all the code has logical sense) in order to reduce the subjectivity in the marking process.

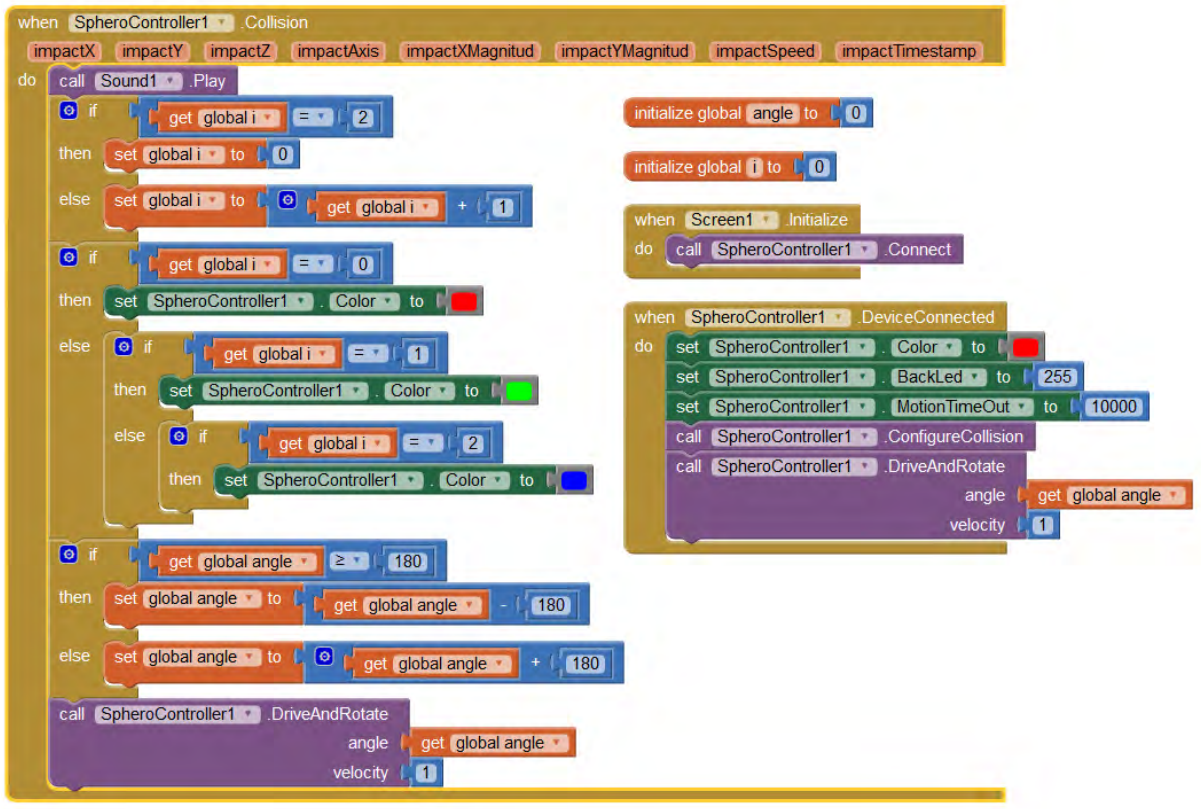
The maximum time available for completing the three exercises was ninety minutes. This time could be freely distributed among the exercises.

From the three exercises (''SquareRoll'', ''RGBLed'' and ''MovingCollision''), we obtained the following data for each student: the time taken to solve each exercise (TM1, TM2 and TM3 indicators), the perception of the difficulty level (DF1, DF2 and DF3 indicators) and the achieved mark (MR1, MR2 and MR3 indicators), as well as the perceived enjoyment level in relation to the use of the development environment (EJ indicator).

Finally, in order to complement and qualify the quantitative results obtained from the experimental tests, the students participating in the experiments answered a survey asking them to describe the impressions about their experience. The survey document is also included in the ZIP package mentioned in Subsection B.

V. RESULTS AND ANALYSIS

Tables 1 and 2 show the values for the set of indicators described in Subsection 4.B, along with the corresponding standard deviations. These values are shown for the three groups of students that have taken part in the experiment:



```
when SpheroController1 .Collision
  impactX  impactY  impactZ  impactAxis  impactXMagnitud  impactYMagnitud  impactSpeed  impactTimestamp
do  call Sound1 .Play
    if    get global i = 2
    then  set global i to 0
    else  set global i to  get global i + 1
    if    get global i = 0
    then  set SpheroController1 . Color to 🟥
    else  if    get global i = 1
          then  set SpheroController1 . Color to 🟩
          else  if    get global i = 2
                then  set SpheroController1 . Color to 🟦
    if    get global angle ≥ 180
    then  set global angle to  get global angle - 180
    else  set global angle to  get global angle + 180
    call SpheroController1 .DriveAndRotate
                                   angle  get global angle
                                velocity  1

initialize global angle to 0

initialize global i to 0

when Screen1 .Initialize
do  call SpheroController1 .Connect

when SpheroController1 .DeviceConnected
do  set SpheroController1 . Color to 🟥
    set SpheroController1 . BackLed to 255
    set SpheroController1 . MotionTimeOut to 10000
    call SpheroController1 .ConfigureCollision
    call SpheroController1 .DriveAndRotate
                           angle  get global angle
                        velocity  1
```

FIGURE 7. "MovingCollision" program developed with the VEDILS authoring tool.

TABLE 1. Values of indicators and standard deviations (student perceptions) for the three groups of students.

	C# group		Sphero Edu group		VEDILS group	
Indicator name	Mean	Std. dev.	Mean	Std. dev.	Mean	Std. dev.
Clarity level (presentation)	3.00	0.55	3.75	0.43	3.45	0.50
Interest level (presentation)	3.55	0.67	3.90	0.30	4.00	0.00
Difficulty level (exercise 1)	2.55	0.50	1.40	0.49	2.55	0.50
Difficulty level (exercise 2)	2.30	0.71	1.15	0.36	2.15	0.73
Difficulty level (exercise 3)	3.70	0.46	3.55	0.50	2.85	0.65
Enjoyment (environment)	2.95	0.59	3.75	0.43	3.55	0.50

TABLE 2. Values of indicators and standard deviations (student learning data) for the three groups of students.

	C# group		Sphero Edu group		VEDILS group	
Indicator name	Mean	Std. dev.	Mean	Std. dev.	Mean	Std. dev.
Spent time (study)	87.75	9.55	127.25	27.18	129.50	16.12
Achieved mark (exercise 1)	1.40	0.49	2.00	0.00	1.85	0.36
Achieved mark (exercise 2)	1.45	0.50	2.00	0.00	2.00	0.00
Achieved mark (exercise 3)	1.05	0.50	1.75	0.43	1.70	0.46
Spent time (exercise 1)	38.50	7.51	9.30	2.28	28.55	5.93
Spent time (exercise 2)	25.55	5.50	8.05	2.04	24.30	4.84
Spent time (exercise 3)	25.95	7.67	27.80	6.06	32.15	5.81

- Learning Scenario I: Students that have participated in the C# OOP course using the *SpherOOP* API and the *Microsoft Visual Studio* IDE (C# group).
- Learning Scenario II: Students that have participated in the visual programming course using the *Sphero Edu* app for mobile devices (*Sphero Edu* group).

TABLE 3. Wilcoxon rank sum test for independent samples (α = 0.05). C# and *sphero edu* groups.

Null hypothesis	p-value	Decision
The distribution of CL is the same between the C# and Sphero Edu groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of IT is the same between the C# and Sphero Edu groups.	*0.165*	*Retain the null hypothesis.*
The distribution of ST is the same between the C# and Sphero Edu groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of MR1 is the same between the C# and Sphero Edu groups.	*0.001*	*Reject the null hypothesis.*
The distribution of MR2 is the same between the C# and Sphero Edu groups.	*0.002*	*Reject the null hypothesis.*
The distribution of MR3 is the same between the C# and Sphero Edu groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of TM1 is the same between the C# and Sphero Edu groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of TM2 is the same between the C# and Sphero Edu groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of TM3 is the same between the C# and Sphero Edu groups.	*0.265*	*Retain the null hypothesis.*
The distribution of DF1 is the same between the C# and Sphero Edu groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of DF2 is the same between the C# and Sphero Edu groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of DF3 is the same between the C# and Sphero Edu groups.	*0.429*	*Retain the null hypothesis.*
The distribution of EJ is the same between the C# and Sphero Edu groups.	*< 0.001*	*Reject the null hypothesis.*

TABLE 4. Wilcoxon rank sum test for independent samples (α = 0.05). C# and VEDILS groups.

Null hypothesis	p-value	Decision
The distribution of CL is the same between the C# and VEDILS groups.	*0.038*	*Reject the null hypothesis.*
The distribution of IT is the same between the C# and VEDILS groups.	*0.060*	*Retain the null hypothesis.*
The distribution of ST is the same between the C# and VEDILS groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of MR1 is the same between the C# and VEDILS groups.	*0.014*	*Reject the null hypothesis.*
The distribution of MR2 is the same between the C# and VEDILS groups.	*0.002*	*Reject the null hypothesis.*
The distribution of MR3 is the same between the C# and VEDILS groups.	*0.001*	*Reject the null hypothesis.*
The distribution of TM1 is the same between the C# and VEDILS groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of TM2 is the same between the C# and VEDILS groups.	*0.738*	*Retain the null hypothesis.*
The distribution of TM3 is the same between the C# and VEDILS groups.	*0.011*	*Reject the null hypothesis.*
The distribution of DF1 is the same between the C# and VEDILS groups.	*1.0*	*Retain the null hypothesis.*
The distribution of DF2 is the same between the C# and VEDILS groups.	*0.547*	*Retain the null hypothesis.*
The distribution of DF3 is the same between the C# and VEDILS groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of EJ is the same between the C# and VEDILS groups.	*0.007*	*Reject the null hypothesis.*

- Learning Scenario III: Students that have participated in the visual programming course using the VEDILS authoring tool (VEDILS group).

Data from Tables 3, 4 and 5 indicate if significant differences have been found when comparing the values of the indicators for the experimental groups.

TABLE 5. Wilcoxon rank sum test for independent samples (α = 0.05). *sphero edu* and VEDILS groups.

Null hypothesis	p-value	Decision
The distribution of CL is the same between the Sphero Edu and VEDILS groups.	*0.108*	*Retain the null hypothesis.*
The distribution of IT is the same between the Sphero Edu and VEDILS groups.	*0.602*	*Retain the null hypothesis.*
The distribution of ST is the same between the Sphero Edu and VEDILS groups.	*0.678*	*Retain the null hypothesis.*
The distribution of MR1 is the same between the Sphero Edu and VEDILS groups.	*0.429*	*Retain the null hypothesis.*
The distribution of MR2 is the same between the Sphero Edu and VEDILS groups.	*1.0*	*Retain the null hypothesis.*
The distribution of MR3 is the same between the Sphero Edu and VEDILS groups.	*0.799*	*Retain the null hypothesis.*
The distribution of TM1 is the same between the Sphero Edu and VEDILS groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of TM2 is the same between the Sphero Edu and VEDILS groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of TM3 is the same between the Sphero Edu and VEDILS groups.	*0.024*	*Reject the null hypothesis.*
The distribution of DF1 is the same between the Sphero Edu and VEDILS groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of DF2 is the same between the Sphero Edu and VEDILS groups.	*< 0.001*	*Reject the null hypothesis.*
The distribution of DF3 is the same between the Sphero Edu and VEDILS groups.	*0.003*	*Reject the null hypothesis.*
The distribution of EJ is the same between the Sphero Edu and VEDILS groups.	*0.289*	*Retain the null hypothesis.*

From the obtained results, we can observe that, for the clarity level indicator (Table 1), the *Sphero Edu* group and the VEDILS group achieve higher values than the C# group. This result could be expected due to the characteristics of VPLs, that are more descriptive thanks to their visual notation and, consequently, facilitate the exposition of the contents about programming in these languages. Moreover, data from Tables 3 and 4 show the existence of significant differences in the *CL* indicator.

However, data from Tables 3 and 4 do not show the existence of significant differences in relation to the interest level indicator values. This can be explained by the fact that the three groups of students knew that they were going to develop programs for controlling a Sphero robot, which is an important factor of interest. Especially, when the participants in the experimental tests were students that did not have a previous contact with robot programming.

Anyway, a higher level of interest can be appreciated for the Sphero Edu and VEDILS groups compared to the C# group (Table 1), since the nature of VPLs also contributes to the interest in the exposition of the corresponding courses.

The spent time (study) indicator (Table 2) is smaller for the students of the C# group, and greater but similar for the students of the Sphero Edu and VEDILS groups. Data from Tables 3 and 4 show the existence of significant differences between the Sphero Edu and C# groups, and between the C# and VEDILS groups for the *ST* indicator. VPLs were new for the students and, therefore, required a greater time for their study. However, the students of the C# group already had a knowledge on imperative programing in C language, on which they only had to learn what C# adds, which is mainly the object orientation paradigm.

The achieved marks for the three exercises are lower for the C# group compared to the other ones. Moreover, data from Tables 3 and 4 indicate the existence of significant differences between the Sphero Edu and C# groups, and between the C# and VEDILS groups for *MR1*, *MR2* and *MR3* indicators. Compared to VPLs, the complexity of developing a program in a textual language is greater, due to the possibility of making syntax errors and the use of a more open development environment (on the contrary, a visual language provides a more or less extensive set of programming primitives in the form of graphic blocks), in addition to being less intuitive than a visual language.

It is important to note that the exercise relating to the collision event detection (''MovingCollision'') has obtained the lowest average mark in the three experimental groups. This exercise is the most complex of the three ones. However, those students who have worked in visual environments (*Sphero Edu* app and VEDILS) have achieved a better average mark compared to their classmates who have worked in the textual programming environment.

Although the paradigm of event-based programming is new for all the students, the programming environments based on VPLs simplify considerably the writing of the code that enables the detection of events - the Sphero robot collisions in this case - and the code for the corresponding hander methods. When using the VEDILS authoring tool, the ''ConfigureCollision'' block (Fig. 7) must be included in the visual programming code in order to activate the collision detection. Also, the ''Collision'' block allows to program the corresponding handler method. In the case of the *Sphero Edu* app, the method for enabling and handling the collision detection events is programmed using the ''On Collision'' block.

However, when programming in C# a detailed knowledge about its concrete syntax is necessary for enabling the collision event, writing the corresponding handler method and subscribing the collision event with the handler method name.

As for the time spent completing each exercise (Table 2), the students from the *Sphero Edu* group took less time compared to the other two groups (C# and VEDILS) in carrying out the first two exercises (Tables 3 and 5 indicate the existence of significant differences for *TM1* and *TM2* indicators). However, they spent rather more time doing the third one (*TM3* indicator), that has a greater complexity. Moreover, all the students of the C# group spent all the time available (ninety minutes) for solving the three exercises, but none of them was able to complete all the exercises correctly. This fact can be explained due to the greater complexity of developing programs using a textual language instead of a visual one.

With regard to the difficulty level perceived by the students when solving the three exercises, such level is consistent with their complexity. The second exercise (''RGBLed'') is the simplest one, since the tasks to perform only consist of changing the color of the Sphero main LEDs every second to red, green and blue in sequence, and playing in the mobile device a sound associated to each color.

However, the third exercise is the most complex since it is related to event-based programming, which is a new paradigm for students initiated in imperative programming, as in the present case. In fact, all the three experimental groups - C#, Sphero Edu and VEDILS - perceived a higher difficulty level when doing the third exercise.

The students of the three groups perceived an intermediate difficulty level associated to the resolution of the first exercise (''SquareRoll''). This fact corresponds to reality, since the control algorithm for the straight-line motions and the rotations of the Sphero robot in order to perform a square-shaped path is more complex than the one that changes its color (''RGBLed''). However, the first exercise is simpler than the third one (''MovingCollision''), due to the event-based programming component that the latter includes.

Data from Table 5 indicate the existence of significant differences between the Sphero Edu and VEDILS groups for *TM1*, *TM2*, *TM3*, *DF1*, *DF2* and *DF3* indicators, whereas the existence of significant differences is not shown for the rest of the indicators. This result was predictable since both developing environments are similar to a certain extent, given that they are based on visual programming and, particularly, considering that the experimental tests were circumscribed to the specific context of the Sphero robot programming.

According to the differences in the indicators previously mentioned, it seems reasonable to think that due to the greater development complexity inherent to the VEDILS authoring tool, the students of the VEDILS group took more time to solve the three exercises - ''SquareRoll'', ''RGBLed'' and ''MovingCollision'' - in comparison with the students of the Sphero Edu group. Certainly, the *Sphero Edu* app is a more closed environment than VEDILS, and exclusively oriented to the use of the Sphero robotic ball.

Moreover, in relation to the perceived level of difficulty, the students of the Sphero Edu group indicated a lower level for the first two exercises (*DF1* and *DF2* indicators) compared to the VEDILS group, which can also explained by the reason described in the previous paragraph.

However, the perceived difficulty level when solving the third exercise (*DF3* indicator), which is the most complex, was a bit higher for the *Sphero Edu* group. The difference with the VEDILS group is only 0.7 points, but it is significant according to the corresponding data from Table 5. This result can be explained given the subjective nature of the *DF3* indicator, and the fact that the samples of the *Sphero Edu* and the VEDILS groups are independent. In the context of these experimental tests, the students can compare the perceived difficulty levels when solving their own exercises, but they cannot make comparisons with the exercises corresponding to those learning environments in which they have not taken part.

The enjoyment level indicator in the use of the concrete development environment (Table 1), in contrast with the learning scenario based on the C# language, achieves a higher value for the learning scenario based on the VEDILS authoring tool, and achieves its maximum value for the scenario based on the *Sphero Edu* app. This fact can be easily explained due to the characteristics of VPLs (use of a visual environment, absence of syntactical errors, greater abstraction level, etc.), that make the programming task a more entertaining and intuitive process. In fact, data from Tables 3 and 4 shows the existence of significant differences in the *EJ* indicator between the C# and Sphero Edu groups, and between the C# and VEDILS groups.

In general, a real improvement in the experimental results (i.e. the clarity level perceived by the students when attending the presentations of the courses, the marks achieved and the difficulty levels perceived when solving the programming exercises, as well as the enjoyment level perceived in the use of the development environments) can be appreciated for the Sphero Edu and VEDILS groups in comparison with the C# group.

VI. DISCUSSION

As indicated in Subsection 4.C, the students participating in the experiments answered a survey asking them to describe the impressions about their experience. The most significant responses are inserted in the rest of this section as qualitative data.

The work presented in this paper has introduced the students, in an enjoyable and practical way, to the fields of robot programming, event-based programming, object-oriented programming and visual programming. In this way, the students have been involved in an integrated learning experience in an interdisciplinary environment.

[C#]*This programming style leads to a new way of solving problems, since they can be applied to objects found in the real world (like a robot). This approach is far from the programs to solve equations I am used to.*

[C#] *It has been an interesting activity related to robotics and programming. The fact of sending commands to a physical object (robot) for controlling it makes the activity very interesting for those passionate about robotics.*

[*Sphero Edu*] *It has been an innovative and fun course that has led us, almost without realizing it, to the direct and enjoyable practice without the need for spending endless hours of theoretical study.*

[*Sphero Edu*] *I encourage all the students to take part in similar projects based on this development environment, since it is so easy to learn how to program a robot for performing tasks even without having previous knowledge on computer programming.*

The obtained experimental results show that a visual programming language can also be suitable for robot programming. The main advantages derived from the use of a VPL in this context include a more intuitive programming process and the ease of use of the development environment corresponding to the VPL. In the visual contexts used for this work, the programmer chooses a set of programming blocks after selecting a block category (*Sphero Edu* app) or a component (App Inventor/VEDILS), which is easier than writing a set of statements in a textual programming environment.

[*Sphero Edu*] *I have found the experience interesting and enjoyable. I had never programmed using a visual environment. Really, what surprised me most was the facility for learning to program using a visual language.*

[VEDILS] *The development environment is very easy to use, and the programs can be imported and exported in a fast and simple way. Moreover, it is so intuitive that it is not necessary for a person to have an in-depth knowledge about programming in order to use it and develop a number of applications for different purposes.*

[VEDILS] *After attending this course, several features of the development environment have helped me to deal with computer programming. The use of the block interface allows students to understand programming concepts better, and also helps those with previous concepts to acquire more in-depth knowledge.*

[VEDILS] *After taking part in the activity, I find programming a less complex task than before. I think that the use of the block-based environment is a simple and useful way of getting started in the field of computer programming, since it allows students and teachers to learn a skill used by relatively few people.*

Also, the obtained results and the student's responses show that those who worked with VPLs have enjoyed more the programming process compared to the students who used the textual programming language. Moreover, the students using VPLs were able to understand better the tasks to be performed by the robot when viewing the block diagrams that specified the example programs thanks to the visual notation. However, when understanding a program written using a textual language, the programmer must visualize in his or her mind the meaning of the statements. In this case, a higher level of abstraction is necessary.

[*Sphero Edu*] *The experience of programming in a visual language was enjoyable because it could be seen what the program does, whereas the use of a textual language makes necessary to know the meaning of every programming element, which makes easier to lose track of the program code. The sensation of enjoyment was due to the awareness that the actions executed by the robot could be seen graphically expressed in the visual program that was shown on the tablet screen, without the need to interpret code statements.*

[*Sphero Edu*]*The resolution of the exercises becomes a very convenient process, since you can visualize your idea about the robot behavior - movements, lights and sounds - at any time directly on the (visual) program you are working on.*

In this vein, various studies [54], [55] state that people understand programs developed using VPLs better, since the human visual system is clearly optimized for processing multi-dimensional data. However, textual computer programs are presented in a one-dimensional form.

"The primary advantage of visual programming languages is that they provide direct representations of software structures such as algorithms and data. This is in contrast to

traditional textual languages, where such multi-dimensional structures are encoded into one-dimensional strings (statements) according to some intricate syntax" [55].

Moreover, VPLs abstract many specific and even uncomfortable aspects of programming. They provide the abstraction to the student [5] in order to make the programming task easier, since VPLs only enhance the logic that is directly relevant to the application, and hide more specific aspects of programming such as storage allocation, scope rules for objects or event loops. Also, the use of programming blocks allows to avoid the issues related to concrete syntax, and facilitates the writing process thanks to menu systems for selecting the necessary blocks in every moment.

In our opinion, the learning experience consisting of programming and controlling the robotic ball operation has helped students of university degrees not directly related to the Computer Science discipline, such as Industrial Engineering, to have a better understanding about the usefulness of computer programming.

[C#] *It has been an interesting experience where we have seen that computer programming can be used to operate a physical device, like a robot.*

[C#] *I now see programming as a more useful tool than I initially thought. An Industrial Engineering student must be familiar with robotics at the hardware and software levels. It is rewarding to learn an object-oriented language.*

[C#] *I was already interested in robotics, but after taking part in the activity my interest is now greater. It is exciting to see how an object can move and to be able to give it instructions about the tasks that you want it to perform. Thus, I find computer programming so important and the experience very interesting.*

[*Sphero Edu*] *I have enjoyed the course more than I expected. Sincerely, I thought that it was going to be boring, but I felt attracted by the idea of programming a robot, executing the program and seeing the result. The development environment is very easy to use and anyone with some knowledge of computer science can use it. It is very satisfying to know that the program that you are writing has a specific utility.*

[VEDILS] *Being able to control the Sphero robot using a mobile app developed through simple steps is just one example of the potential that this environment can offer in contexts related to robotics.*

[VEDILS] *It is very interesting to write simple interactive programs in order to know the resources of the mobile device that can be used. The use of this development environment could be an effective method for introducing young people to the world of computer programming due to its graphical interface and its ease of use.*

Moreover, it is important to note that all the students who answered the survey expressed their interest in taking part in future projects related to programming and robotics. This fact underlines the motivating nature of this experience.

Despite both the *Sphero Edu* app and the VEDILS authoring tool are based on the use of VPLs, the latter tool has the advantage of working in a more open environment. Unlike the *Sphero Edu* app, VEDILS allows the use of the set of sensors and gadgets available in an Android device in addition to the Sphero robot, and also allows to perform the programming process in a computer desktop environment, which use may be more comfortable than the tactile interface characteristic of a smartphone or a tablet.

From our experience, we think that the use of VPLs not only has a positive influence on the student's learning process, but it also provides a greater quality to the professor's teaching activity. In fact, the teaching of basic and advanced programming concepts, such as those related to event-based programming, is made easier and improved due to the visual nature of these languages.

[VEDILS] *The experience was very rewarding in relation to the way in which the course contents were explained, and mainly to the eminently visual appearance that the practical examples presented. The course was so interesting since it made us discover a new programming method.*

[VEDILS] *Before taking part in the activity, I saw programming as something boring and unattractive that only had statements and numbers. However, with somebody who teaches you with motivation, in an interactive way, and helps you to think that you can do it, you can learn.*

The visual component of these languages facilitates notably the understanding of a meaningful number of programming concepts that, explained in the context of a textual language, require a greater effort by the instructor so that they can be understood by the students. For example, when introducing the students to C# language, it was necessary to define the concepts of class and object. However, with App Inventor/VEDILS there was no need to explain that dragging a component (class) to the design window, makes the system to create automatically an instance (object) of that component and associate to it the component name followed by an integer, as a default name for the instance.

On the other hand, the portability of the Sphero robot due to its low weight and small size, in addition to its affordable price, as well as the open-source software used [23], [44], [48] and the availability of the *Sphero Edu* [36] and VEDILS [14], [15] development tools, and the accessibility of related documentation [38], [46], [49], allow the replication of the experiments. The portable infrastructure used is, therefore, an important advantage of this work.

In the future, the possibility of using some of the metrics provided by the multidimensional analysis module of VEDILS (e.g. the number of user interface components or programming blocks sorted by type, category, project and user, as well as the number of builds or debug sessions classified by projects and users) could be applied to the analysis of learning processes and the consequent improvement of teaching strategies in the context of robot programming through the use of the VEDILS authoring tool).

Finally, our work suggests the convenience that, in addition to the use of traditional textual programming languages, industrial robots can also be programmed using VPLs.

Thus, those people that are not IT professionals but work with robots (e.g. in a production plant) would see their daily tasks facilitated and streamlined, and even their work would become more enjoyable. The potential of VPLs in the context of industrial robots is also considered in [32] and [33]. The fact that a VPL may not be as flexible as a textual language is not necessarily a disadvantage when the tasks to be performed by the robots in an industrial environment are not excessively complex, which is usually the case.

A. THREADS TO VALIDITY AND LIMITATIONS

In order to allow the reader to evaluate the confidence that can be placed on the results of this work, it is necessary to identify potential issues of validity and bias that may occur during the design and the development of the study.

An internal validity threat of this work consists of its condition of pilot study, carried out through experimental tests performed with not very large groups (twenty students for each group), that is a first step from which more in-depth studies can be conducted. Anyway, the obtained results and the students' responses underline the suitability of VPLs for robot programming.

Unfortunately, although the participants were given an incentive consisting of improving their final mark in the course (*Fundamentals of Computer Science*), the total number was not greater due to the high level of difficulty of the Industrial Engineering studies, which require a lot of time, effort and a great dedication.

Furthermore, since it was necessary for the students to acquire the necessary knowledge of the C programming language before performing the experimental tests, these tests had to be carried out at the end of the course. Thus, the examination period was very near and, consequently, the students had even less time available.

Also, the number of learning scenarios - one based on a textual programming language and two based on VPLs - could be considered an internal validity threat themselves. Certainly, a greater number of scenarios (for example, three based on textual languages and another three based on visual languages) would have allowed us to obtain more meaningful results. Obviously, first of all we would need a great number of participants in order to have an appropriate sample size in each learning scenario.

Moreover, the fact that the samples were composed only by students of Industrial of Engineering can be viewed as an external validity threat, since we cannot assure that the obtained results can be generalized to students of university grades in disciplines very different from Industrial Engineering.

Another validity threat is related to the use of variables for measuring the perceptions of the students (clarity and interest level when attending the exposition of the course contents about the specific programming language, difficulty level when solving the exercises and enjoyment level when using the specific development environment), due to their subjective nature.

In fact, as explained in previous section, data from Table 5 shows that the values for the perceived level of difficulty when solving exercises 1 and 2 (*DF1* and *DF2* indicators) are lower for the *Sphero Edu* group than the corresponding ones for the VEDILS group, whereas the values for *DF3* indicator (exercise 3) are higher for the *Sphero Edu* group. However, the *Sphero Edu* app is a simpler and more easy to use environment than VEDILS, since it is exclusively oriented to the use of the Sphero robot.

VII. CONCLUSION

In this work, a number of functionalities have been added to a visual programming-based authoring tool (VEDILS) in order to make it capable of controlling a commercial mobile robot (Sphero). This tool, along with an Android app (*Sphero Edu*) and a C# API (*SpherOOP*) have been used as development environments where programs for controlling the Sphero robot operation have been written and tested. A Bluetooth link allows the robot to communicate wirelessly with the hardware (computer or Android device according to the specific development environment in use).

A comparison among these three development environments has been made through the analysis of empirical data collected from a set of experimental tests about robot programming carried out by a sample of Industrial Engineering students at the University of Cádiz (Spain). From the analysis of the results presented in this work, we can conclude that a visual programming language can also be suitable for robot programming. Furthermore, the main advantages derived from the use of a VPL in such context, when compared to a textual programming language, include a more intuitive and enjoyable programming process, a greater ease of use of the development environment and a better understanding of the tasks to be performed by the robot when viewing the block diagram that specifies a program.

Furthermore, the educational approach proposed in this work has familiarized the students with visual programming environments based on block languages, object-oriented programming and event-based programming. It also has introduced them to the fields of robotics and robot programming from an eminently practical point of view, thus leading to an enriching learning experience in an interdisciplinary context.

ACKNOWLEDGMENT

Illustration 1 is provided by courtesy of Sphero.

REFERENCES

[1] E. Lahtinen, K. Ala-Mutka, and H. Järvinen, "A study of the difficulties of novice programmers," in *Proc. 10th Annu. Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, Caparica, Portugal, 2005, pp. 14–18, doi: 10.1145/1151954.1067453.

[2] I. Milne and G. Rowe, "Difficulties in learning and teaching programming—Views of students and tutors," *Educ. Inf. Technol.*, vol. 7, no. 1, pp. 55–66, 2002, doi: 10.1023/A:1015362608943.

[3] A. Gomes and A. J. Mendes, "Learning to program—Difficulties and solutions," in *Proc. Int. Conf. Eng. Educ. (ICEE)*, Coimbra, Portugal, 2007.

[4] M. Erwig, K. Smeltzer, and X. Wang, "What is a visual language?" *J. Vis. Lang., Comput.*, vol. 38, pp. 9–17, Feb. 2017, doi: 10.1016/j.jvlc.2016.10.005.

[5] R. Navarro-Prieto and J. J. Cañas, "Are visual programming languages better? The role of imagery in program comprehension," *Int. J. Hum.-Comput. Stud.*, vol. 54, no. 6, pp. 799–829, 2001, doi: 10.1006/ijhc.2000.0465.

[6] M. Khademi, M. Haghshenas, and H. Kabir, "A review on authoring tools," in *Proc. 5th Int. Conf. Distance Learn. Educ. (ICDLE)*, Singapore, 2011, pp. 45–49.

[7] M. Resnick, "Sowing the seeds for a more creative society," *Learn. Lead. Technol.*, vol. 35, no. 4, pp. 18–22, 2008.

[8] Lifelong Kindergarten Group. (Apr. 16, 2018). *Scratch—Imagine, Program, Share*. [Online]. Available: https://scratch.mit.edu/

[9] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, "From scratch to 'real' programming," *ACM Trans. Comput. Educ.*, vol. 14, no. 4, 2015, Art. no. 25, doi: 10.1145/2677087.

[10] D. Wolber, H. Abelson, E. Spertus, and L. Looney, *App Inventor 2: Create Your Own Android Apps*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015.

[11] Y. C. Hsu, K. Rice, and L. Dawley, "Empowering educators with Google's Android app inventor: An online workshop in mobile App design," *Brit. J. Educ. Technol.*, vol. 43, no. 1, pp. E1–E5, 2012, doi: 10.1111/j.1467-8535.2011.01241.x.

[12] (2017). *MIT App Inventor—Explore MIT App Inventor*. Accessed: Apr. 16, 2018. [Online]. Available: http://appinventor.mit.edu/explore/

[13] M. S. Horn, E. T. Solovey, R. J. Crouser, and R. J. Jacob, "Comparing the use of tangible and graphical programming languages for informal science education," in *Proc. 27th Conf. Hum. Factors Comput. Syst. (CHI)*, Boston, MA, USA, 2009, pp. 975–984, doi: 10.1145/1518701.1518851.

[14] VEDILS Team. (2018). *(VEDILS—Visual Environment for Designing Interactive Learning Scenarios)*. Accessed: Apr. 16, 2018. [Online]. Available: http://vedils.uca.es/web/

[15] J. M. Mota, I. Ruiz-Rube, J. M. Dodero, and M. Figueiredo, "Visual environment for designing interactive learning scenarios with augmented reality," in *Proc. 12th Int. Conf. Mobile Learn.*, Algarve, Portugal, 2016, pp. 67–74.

[16] A. Balderas, I. Ruíz-Rube, J. M. Mota, J. M. Dodero, and M. Palomo-Duarte, "A development environment to customize assessment through students interaction with multimodal applications," in *Proc. Int. Conf. Technol. Ecosyst. Enhancing Multiculturality (TEEM)*, Salamanca, Spain, 2016, pp. 1043–1048, doi: 10.1145/3012430.3012644.

[17] J. Arlegui, E. Menegatti, M. Moro, and A. Pina, "Robotics, computer science curricula and interdisciplinary activities," in *Proc. 1st Int. Conf. Simulation, Modeling, Program. Auton. Robots (SIMPAR)*, Venice, Italy, 2008, pp. 10–21.

[18] S. Kurkovsky, "Mobile computing and robotics in one course: Why not?" in *Proc. 18th Annu. Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, Canterbury, U.K., 2013, pp. 64–69, doi: 10.1145/2462476.2465584.

[19] S. Kurkovsky, "Interdisciplinary connections in a mobile computing and robotics course," in *Proc. 19th Annu. Conf. Innov. Technol. Comput. Sci. Educ. (ITiCSE)*, Uppsala, Sweden, 2014, pp. 309–314, doi: 10.1145/2591708.2591735.

[20] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, "Graphical programming environments for educational robots: Open Roberta—Yet another one?" in *Proc. IEEE Int. Symp. Multimedia (ISM)*, Taichung, Taiwan, Dec. 2014, pp. 381–386, doi: 10.1109/ISM.2014.24.

[21] Sphero. (2017). *Sphero 2.0—Playtime Just Got Real*. Accessed: Apr. 8, 2018. [Online]. Available: https://www.sphero.com/sphero/

[22] Sphero. (2017). *Sphero Docs—What is Sphero*. Accessed: Apr. 8, 2018. [Online]. Available: http://sdk.sphero.com/sphero-robot-basics/what-is-sphero/

[23] J. M. R. Corral *et al.*, "Application of robot programming to the teaching of object-oriented computer languages," *Int. J. Eng. Educ.*, vol. 32, no. 4, pp. 1823–1832, 2016.

[24] T. Faison, *Event-Based Programming: Taking Events to the Limit*. Breinigsville, PA, USA: Apress, 2006, doi: 10.1007/978-1-4302-0156-4.

[25] T. Booth and S. Stumpf, "End-user experiences of visual and textual programming environments for Arduino," in *End-User Development* (Lecture Notes in Computer Science), vol. 7897, Y. Dittrich, M. Burnett, A. Mørch, and D. Redmiles, Eds. Berlin, Germany: Springer-Verlag, 2013, pp. 25–39, doi: 10.1007/978-3-642-38706-7_4.

[26] D. Weintrop and U. Wilensky, "To block or not to block, that is the question: Students' perceptions of blocks-based programming," in *Proc. 14th Int. Conf. Interact. Design Children (IDC)*, Boston, MA, USA, 2015, pp. 199–208, doi: 10.1145/2771839.2771860.

[27] T. W. Price and T. Barnes, "Comparing textual and block interfaces in a novice programming environment," in *Proc. 11th Annu. ACM Int. Comput. Educ. Res. Conf. (ICER)*, Omaha, NE, USA, 2015, pp. 91–99, doi: 10.1145/2787622.2787712.

[28] D. Weintrop and U. Wilensky, "Comparing block-based and text-based programming in high school computer science classrooms," *ACM Trans. Comput. Educ.*, vol. 18, no. 1, pp. 1–25, 2017, doi: 10.1145/3089799.

[29] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Learnable programming: Blocks and beyond," *Commun. ACM*, vol. 60, no. 6, pp. 72–80, 2017, doi: 10.1145/3015455.

[30] J. P. Diprose, B. A. MacDonald, and J. G. Hosking, "Ruru: A spatial and interactive visual programming language for novice robot programming," in *Proc. IEEE Symp. Visual Lang. Hum.-Centric Comput. (VL/HCC)*, Pittsburgh, PA, USA, Sep. 2011, pp. 25–32, doi: 10.1109/VLHCC.2011.6070374.

[31] F. Riedo, M. Chevalier, S. Magnenat, and F. Mondada, "Thymio II, a robot that grows wiser with children," in *Proc. IEEE Workshop Adv. Robot. Social Impacts (ARSO)*, Tokyo, Japan, 2013, pp. 187–193, doi: 10.1109/ARSO.2013.6705527.

[32] D. Weintrop, D. C. Shepherd, P. Francis, and D. Franklin, "Blocky goes to work: Block-based programming for industrial robots," in *Proc. IEEE Blocks Beyond Workshop (B&B)*, Raleigh, NC, USA, Oct. 2017, pp. 29–36, doi: 10.1109/BLOCKS.2017.8120406.

[33] D. Weintrop *et al.*, "Evaluating CoBlox: A comparative study of robotics programming environments for adult novices," in *Proc. CHI Conf. Hum. Factors Comput. Syst. (CHI)*, Montreal QC, Canada, 2018, doi: 10.1145/3173574.3173940.

[34] F. Barreto and V. Benitti, "Exploring the educational potential of robotics in schools: A systematic review," *Comput. Educ.*, vol. 58, no. 3, pp. 978–988, 2012, doi: 10.1016/j.compedu.2011.10.006.

[35] D. Alimisis, "Educational robotics: Open questions and new challenges," *Themes Sci., Technol. Educ.*, vol. 6, no. 1, pp. 63–71, 2013.

[36] Sphero. (2017). *Sphero Edu*. Accessed: Apr. 16, 2018. [Online]. Available: https://edu.sphero.com/

[37] *Danish Business Research Academy (DEA/Danmarks Erhvervsforskningsakademi), and Danish Forum for Business Education (FBE)*, Thinking Across Disciplines—Interdisciplinarity Res. Educ., Copenhagen, Denmark, 2008.

[38] Sphero. (2017). *Sphero Edu—Learning is Evolving. Get on the Ball*. Accessed: Apr. 8, 2018. [Online]. Available: http://www.sphero.com/education/

[39] H. Zhang, Ed. *Climbing and Walking Robots: Towards New Applications*. Vienna, Austria: InTech, 2007, ch. 11, pp. 235–256, doi: 10.5772/47.

[40] Bluetooth SIG, Inc. (2018). *Specifications—Bluetooth Technology Website*. Accessed: Apr. 8, 2018. [Online]. Available: https://www.bluetooth.org/en-us/specification/

[41] A. S. Huang and L. Rudolph, *Bluetooth Essentials for Programmers*. Cambridge, U.K.: Cambridge Univ. Press, 2007, doi: 10.1017/CBO9780511546976.

[42] GitHub, Inc. (2017). *Sphero Sensor Streaming*. Accessed: Apr. 8, 2018. [Online]. Available: https://github.com/orbotix/Sphero-iOS-SDK/tree/master/samples/SensorStreaming/

[43] M. Pagel and D. Carlson, "Ambient control: A mobile framework for dynamically remixing the Internet of Things," in *Proc. IEEE 16th Int. Symp. World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Boston, MA, USA, Jun. 2015, pp. 1–9, doi: 10.1109/WoWMoM.2015.7158143.

[44] T. Bladh. (2013). *Balls Out Fun With the Sphero and .NET*. Accessed: Apr. 8, 2018. [Online]. Available: http://thomasbladh.com/2013/01/01/balls-out-fun-with-the-sphero/

[45] GitHub, Inc. (2013). *Orbotix—Developer Resources*. Accessed: Apr. 16, 2018. [Online]. Available: https://github.com/orbotix/DeveloperResources/zipball/master/

[46] Sphero. (2017). *Sphero Docs—API Quick Reference*. Accessed: Apr. 16, 2018. [Online]. Available: https://sdk.sphero.com/api-reference/api-quick-reference/

[47] F. Weichert, D. Bachmann, B. Rudak, and D. Fisseler, "Analysis of the accuracy and robustness of the leap motion controller," *Sensors*, vol. 13, no. 5, pp. 6380–6393, 2013, doi: 10.3390/s130506380.

[48] GitHub, Inc. 2018. *Sphero Android SDK*. Accessed: Apr. 8, 2018. [Online]. Available: https://github.com/orbotix/Sphero-Android-SDK/

[49] Sphero. (2017). *Sphero Docs—Convenience Robot*. Accessed: Apr. 8, 2018. [Online]. Available: http://sdk.sphero.com/sdk-documentation/convenience-robot/

[50] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 1988.
[51] Microsoft Corporation. (2012). *C# Language Specification, Version 5.0.* Accessed: Apr. 8, 2018. [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=7029
[52] D. Clark, *Beginning C# Object-Oriented Programming*, 2nd ed. New York, NY, USA: Apress, 2013.
[53] M. Michaelis and E. Lipper, *Essential C# 6.0.* Ann Arbor, MI, USA: Addison-Wesley, 2016.
[54] B. A. Myers, "Taxonomies of visual programming and program visualization," *J. Vis. Lang., Comput.*, vol. 1, no. 1, pp. 97–123, 1990, doi: 10.1016/S1045-926X(05)80036-9.
[55] P. T. Cox and T. J. Smedley, "Building environments for visual programming of robots by demonstration," *J. Vis. Lang., Comput.*, vol. 11, no. 5, pp. 549–571, 2000, doi: 10.1006/jvlc.2000.0175.

JOSÉ MARÍA RODRÍGUEZ CORRAL received the master's degree in computer engineering and the Ph.D. degree from the University of Seville, Spain, in 1993 and 2002, respectively. From 1993 to 1995, he worked on robot control with the Robotics and Computer Technology Research Group, University of Seville. He was an Associate Lecturer with the University of Cádiz, Spain, from 1995 to 1998, where he is currently an Associate Professor of computer languages and systems and also a member of the Applied Robotics Group. He has authored various papers and research reports on computer architecture. His research interests include engineering education, robotics, and bus emulation.

IVÁN RUÍZ-RUBE received the master's degree in software engineering and technology from the University of Seville and the Ph.D. degree from the University of Cádiz. He was a Software Engineer for consulting companies, such as Everis Spain S.L. and Sadiel S.A. He is currently an Associate Lecturer with the University of Cádiz. His current research interests include technology-enhanced learning, software process improvement, and linked open data. He has published several papers in these fields.

ANTÓN CIVIT BALCELLS received the master's degree in physics (electronics) and the Ph.D. degree from the University of Seville, Spain, in 1984 and 1987, respectively. After working for several months with Hewlett–Packard, he joined the University of Seville, where he is currently a Full Professor of computer architecture and also the Director of the Robotics and Computer Technology Research Group. He has authored various papers and research reports on computer architecture, rehabilitation technology, and robotics. His research interests include advanced wheelchairs, robotics, and real-time architectures.

JOSÉ MIGUEL MOTA-MACÍAS received the master's degree in computer science with the Universitat Oberta de Catalunya, Spain. He is currently an Associate Lecturer with the University of Cádiz, Spain. He is also a Ph.D. Candidate Researcher in technology-enhanced learning. His current research interests include mobile learning, augmented reality, and learning analytics.

ARTURO MORGADO-ESTÉVEZ received the master's degree in industrial organization engineering and the Ph.D. degree from the University of Cádiz, Spain, in 1997 and 2003, respectively. After working on ASIC design with the Microelectronics Research Group, University of Cádiz, from 1989 to 1998, and also with the Robotics and Computer Technology Group, University of Seville, from 1998 to 2010. He has been an Associate Professor of computer architecture with the University of Cádiz, since 1991, where he is currently the Director of the Applied Robotics Group. His research interests include robotics and life-inspired systems. He has authored various papers on computer architecture.

JUAN MANUEL DODERO received the master's degree in computer science from the Polytechnic University of Madrid and the Ph.D. degree from the Carlos III University of Madrid. He was an Associate Lecturer with the Carlos III University of Madrid and was also a R&D Engineer for Intelligent Software Components S.A. He is currently an Associate Professor with the University of Cádiz, Spain. His current research interests include web science and engineering and technology-enhanced learning. He has co-authored numerous contributions in journals and research conferences.