applied sciences

MDPI

Article

A ROS-Based Open Tool for Intelligent Robotics Education

José M. Cañas [1,*], Eduardo Perdices [2], Lía García-Pérez [3] and Jesús Fernández-Conde [1]

[1] Department of Telematic Systems and Computation, Rey Juan Carlos University, Fuenlabrada, 28942 Madrid, Spain; jesus.fernandez@urjc.es

[2] JdeRobot Organization, Alcorcón, 28922 Madrid, Spain; eperdices@gsyc.urjc.es

[3] Industrial Engineering Department, Francisco de Vitoria University, Pozuelo de Alarcón, 28223 Madrid, Spain; lia.garcia@ufv.es

* Correspondence: josemaria.plaza@urjc.es; Tel.: +34-914-888-755

Received: 15 September 2020; Accepted: 19 October 2020; Published: 22 October 2020

Abstract: This paper presents an open-access platform for practical learning of intelligent robotics in engineering degrees: Robotics-Academy. It comprises a collection of exercises including recent service robot applications in real life, with different robots such as autonomous cars, drones or vacuum cleaners. It uses Robot Operating System (ROS) middleware, the de facto standard in robot programming, the 3D Gazebo simulator and the Python programming language. For each exercise, a software template has been developed, performing all the auxiliary tasks such as the graphical interface, connection to the sensors and actuators, timing of the code, etc. This also hosts the student's code. Using this template, the student just focuses on the robot intelligence (for instance, perception and control algorithms) without wasting time on auxiliary details which have little educational value. The templates are coded as ROS nodes or as Jupyter Notebooks ready to use in the web browser. Reference solutions for illustrative purposes and automatic assessment tools for gamification have also been developed. An introductory course to intelligent robotics has been elaborated and its contents are available and ready to use at Robotics-Academy, including reactive behaviors, path planning, local/global navigation, and self-localization algorithms. Robotics-Academy provides a valuable complement to master classes in blended learning, massive online open courses (MOOCs) and online video courses, devoted to addressing theoretical content. This open educational tool connects that theory with practical robot applications and is suitable to be used in distance education. Robotics-Academy has been successfully used in several subjects on undergraduate and master's degree engineering courses, in addition to a pre-university pilot course.

Keywords: intelligent robotics; engineering education; distance learning; open educational tool

1. Introduction

An increasing number of robotic applications are available to the general public. Beyond the classic industrial applications and automobile assembly processes, robots are used today, for example, in food packaging or in warehouse logistics. Robotic vacuum cleaners have been an unprecedented sales breakthrough, using autonomous robots to successfully solve a real life need at homes. Cars are also increasingly incorporating robotic technologies such as auto-parking or even autonomous driver-assistance systems. The big automakers have encouraged these new technologies, achieving advanced prototypes of autonomous cars. Software companies like Google or Apple have been at the forefront of these developments. In addition, applications with aerial robots (drones) are also rapidly growing in number.

All the robotic applications that reach the general public have software inside and much of their intelligence and value lies in that software. Typically, this software has several layers (such as drivers, middleware and application layers) and has specific requirements different from those required in other fields: real-time operation, robustness, reliability, distributed nature and heterogeneous hardware. In addition, graphical interfaces are often used for debugging, but seldom at runtime.

The growth of robotics technology has created the need to train professionals in the sector, who can take the current limits further and produce new robotics applications to serve common people. Robotics is a cross-disciplinary field involving many technologies: electronics, mechanics, computer science, telecommunications, etc. At universities, robotics subjects and programs are traditionally taught at schools of mechanical engineering, electrical engineering and computer science [1,2], both on degree courses and post-graduate courses.

Prestigious associations such as ACM, IEEE-CS, IEEE-SMC and IEEE-RA consider robotics to be one of the fundamental knowledge areas in computer science and engineering studies, especially in the field of intelligent systems [3]. The leading American universities in technology (Carnegie Mellon University, Stanford, Massachusetts Institute of Technology, Georgia Institute of Technology, etc.) include graduate and post-graduate studies in robotics and related fields, such as computer vision and artificial intelligence. There are many minor, bachelor, master and Ph.D. programs in robotics.

Due to its cross-disciplinary nature, robots can be approached from many perspectives. For instance, as a set of sensors and actuators with intelligent software placed in between. Following this approach, the goal of the present work is to increase the quality of robotics teaching proposing a new open educational resource for higher education named Robotics-Academy. This focuses on software for robots as the key element for their intelligence, more on the algorithms rather than the middleware. It has been designed especially for students learning perception, planning and control algorithms common in robotics syllabuses of university courses (12–14 weeks).

The remainder of this paper is organized as follows: the second section provides an overview of robotics teaching at university level and reviews several teaching tools used by the community. The third section explains the general design of the proposed teaching suite and several key decisions that have guided its development. In Section 4, we describe an introductory Intelligent Robotics course, including diverse practical exercises available in the framework. The fifth section presents results of courses in which the framework has been used and student evaluations. The last section summarizes the main conclusions and future lines of work.

2. Robotics Teaching in Higher Education

There are two large trends in teaching robotics content at university: subjects focused on industrial robots, arms and manipulators [4–6] and subjects based on mobile robots. In the first case, control techniques, inverse and kinematics or trajectory calculations are usually addressed. In the second case, the techniques generally explained are local and global navigation, position control (avoiding obstacles), perception, self-location, etc. This dichotomy of contents is also reflected in the robotic platforms that are used in the exercises.

Both types of robotic subject have a notably practical character. Student interaction with robots facilitates the learning and assimilation of the theoretical concepts, algorithms and techniques [7]. Exercises promote the paradigm of *learning through doing*, or active learning. They usually take place within a specific robotic framework, using a teaching tool where robot behavior is programmed using a certain language. A great diversity of the hardware platforms is used to support the experimental component of robotics courses. Virtual [7–9] and remote laboratories [10] are frequent in the literature. In terms of programming environments, according to Esposito [11], MATLAB (62%) and C (52%) remain the dominant choices, with a minority using free open-source packages such as the Robot Operating System (ROS) (28%) or OpenCV (17%).

One of the most commonly used teaching suites for practice sessions is MATLAB, with its own language, sometimes accompanied by the Simulink package. For example, Gil et al. [12] describe the MATLAB ARTE (A Robotics Toolbox for Education) toolbox, oriented towards manipulation and have used it in their courses including 3D visualization and arm programming with an industrial language.

Aliane [2] uses MATLAB and Simulink to present students' practical sessions on trajectory calculation and a Selective Compliant Articulated Robot Arm (SCARA) manipulator control. Gonzalez et al. [13] have developed the Robot Motion Toolbox which allows mobile robot planning, navigation and control, and includes its own simulator. Corke [14] developed the Robotics Toolbox to program mobile robots, arms and robots with vision [15].

The Mobile Robot Interactive Tool [16] is a teaching platform for mobile robots. It is a two-dimensional environment and is made with SysQuake (similar to MATLAB). It allows the testing and adjustment of parameters of different navigation solutions integrated in the tool (Visibility Graph, Generalized Voronoi Diagram, Wavefront, Bug, Visbug, etc.) and the robot kinematics. They use the Tritton or the PeopleBot robots to follow a trajectory. This platform together with MATLAB is used on the courses organized by Berenguel et al. [3], providing a mixed profile covering both manipulators (Scorbot-ER Plus V) and mobile robots (LEGO, PeopleBot).

Fabregas et al. recently presented Robots Formation Control Platform [9], a teaching suite oriented towards controlling groups of mobile robots and their movement in formations. Rather than focusing on behavior programming, it allows the user to select some of the Vector Field Histogram control algorithms already programmed within the suite (VFH, VFH+, VFH*), configure them in different ways, decide whether there are obstacles or not, choose the desired formation or the robot kinematic model and view the resulting trajectories.

Guyot et al. [17] proposed a complete teaching suite with the Webots simulator for a small mobile robot, the e-puck. This environment includes multi-level exercises, from novice to the intermediate, advanced or expert. The C programming language is used for some of the exercises developed in the suite.

The SyRoTek teaching platform [10] provides web access to 13 small real mobile robots operating in an enclosed space, the Arena. This remote laboratory was used by more than 80 students from the Czech Republic and Argentina in exercises about robot exploration, navigation and map building. It uses Player/Stage framework and ROS interfaces. Farias et al. [18] describe another relevant teaching platform which combines several Khepera IV physical robots and the use of V-REP simulator to teach control of mobile robots. Some teaching experiments inside their platform such as Position Control, Trajectory Tracking, Path Following and Obstacle Avoidance are also reported.

An interesting teaching suite for robotics in computer degrees is Tekkotsu [1]. Initially designed for the Aibo robot, it has been made cross-platform and includes several libraries with already programmed functionality (handling of three-dimensional information, navigation, etc.). The approach focuses on integration and understanding existing code rather than developing a new basic version of an algorithm.

TRS [19] is a free software robotic framework for postgraduate students. It is based on the V-REP simulator, which is multiplatform and has been used in a robotics course at Liege University and Aalborg University Copenhagen. It has outstandingly quick and easy installation. It enables students to program control, navigation or manipulation algorithms in Python or MATLAB code.

In the last five years ROS middleware has become the de facto standard in the robotics research community [20–23] and has also increased its presence in education [24–27]. Some of these works combine or compare ROS and MATLAB [28–30].

In addition, digitalization is changing how knowledge is created, consumed and taught. Universities increasingly offer massive online open courses (MOOC) exploring new educational possibilities such as e-learning. In particular, universities such as Stanford, MIT, Harvard have led initiatives in this direction since 2011. They have succeeded in promoting open learning, although MOOCs have also been criticized for their high drop-out rates and behaviorist pedagogical approach. Some works propose their redesign towards gamification to overcome some of their limitations [31]. The robotics field has also been influenced by this movement and there are increasing numbers of robotic courses of this style [32], like Artificial Intelligence for Robotics (https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373) from Stanford University (Udacity, Sebastian Thrun), Autonomous Navigation for Flying Robots (https://www.edx.org/course/autonomous-navigation-flying-robots-tumx-autonavx-0) from the

Technical University of Munich or Autonomous Mobile Robots (https://www.edx.org/course/autonomous-mobile-robots-ethx-amrx-1) from the ETH Zurich. Another example is the MOOC on the Art of Grasping and Manipulation in Robotics [33] from the University of Siena.

This trend in web-based and practical educational frameworks also appears in other fields. For instance, Google Colaboratory (https://colab.research.google.com) provides a simple-to-use infrastructure where the students can learn deep learning using Jupyter Notebooks from their web browser which are run on Google servers. No installation by the students is required.

In robotics this is complemented by usage of robotic simulators through a web interface. For example, one important online teaching initiative is Robot Ignite Academy from The Construct [34]. It is based on ROS, Gazebo and Jupyter, and provides several short courses. Another is robotbenchmark (https://robotbenchmark.net) from Cyberbotics. It is open, based on Webots simulator and provides 12 Python exercises with several robots such as Thymio II and Nao humanoid.

Another noteworthy ROS-based initiative is Robot Programming Network [35–37]. This extends existing remote robot laboratories with the flexibility and power of writing ROS code in a Web browser and running it in the remote robot on the server side with a single click. It uses Moodle, VirtualBox, html, and Robot Web Tools as its underlying infrastructure.

Table 1 summarizes the features of the main reviewed frameworks for teaching intelligent robotics. Not all of them are compatible with ROS nor support both simulated and physical robots. Only the Robotics-Academy platform proposed in this article provides automatic evaluators.

Table 1. Features of main frameworks for teaching intelligent robotics.

Educative Platform	Programming Language	Supported Robots	Physical Robots	Simulator	Open Source	ROS	Evaluators
Mobile Robot Interactive Tool	SysQuake (~MATLAB)	Three	No	No	~Yes	No	No
Robots Formation Control Platform	~parameters	Moway	Yes	RFC SIM	No	No	No
Guyot et al.	C	e-puck	No	Webots	~Yes	No	No
SyRoTek	C/C++	S1R	Remote	Stage	No	Yes	No
Farias et al.	C/C++	KepheraIV	Remote	V-Rep	No	No	No
TRS	Python/ MATLAB	Several	No	V-Rep	Yes	No	No
Tekkotsu	C++	Several	Yes	Mirage	Yes	No	No
Robot Ignite Academy	Python	Several	No	Gazebo	No	Yes	No
robotbenchmark	Python	ThymioII, Nao	No	Webots	Yes	No	No
Robot Programming Network	Python/ Blockly	Several	Remote	Stage Webots	No	Yes	No
Robotics-Academy	Python	Several	Yes	Gazebo	Yes	Yes	Yes

3. Robotics-Academy Design

The presented educational resource consists of a set of independent exercises. Each exercise proposes a specific robotics problem (typically an autonomous behavior) and the student is intended to develop the intelligence of the robot to solve it. This suite is an evolution from one previously designed [38,39], and its main components are listed below:

- Python programming language
- ROS middleware
- Gazebo simulator
- A code template per exercise (Python ROS node/Jupyter Notebook)
- Automatic evaluator for each exercise

Python has been chosen as programming language because it has a learning curve that is not as steep as C++ and at the same time has the power of object-oriented programming. Python has already been used in teaching robotics several times [40].

The proposed teaching suite encourages the use of *standard libraries* in the field, such as OpenCV for image processing, MoveIt! for robot arms, and Open Motion Planning Library (OMPL) for path planning, etc. In this way, students become familiar with current state-of-the-art libraries that they may encounter on their future professional careers.

Each exercise in Robotics-Academy has three elements, as shown in Figure 1. First, on the lower layer is the robot itself which will perform tasks in a certain environment. It can be either simulated or real. Second, on the intermediate layer there are drivers that give the software access to the sensors and actuators of the robot. Both the robot and the drivers can be reused in different exercises. Third, on the upper layer is the robot application, the academic application in this case. This contains the robot intelligence and analyzes the sensor data and makes decisions or even performs the planning if necessary.

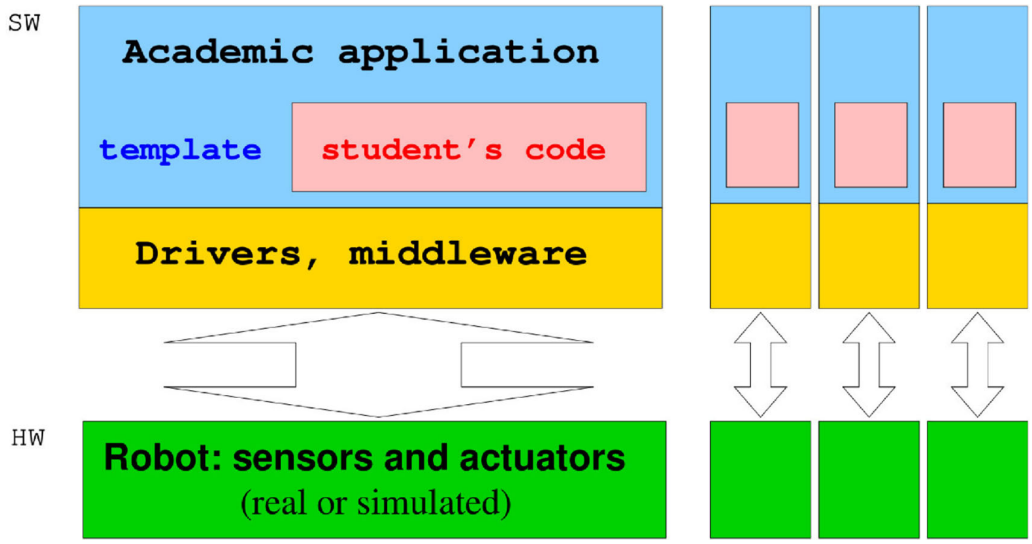

Figure 1. Design of exercises in Robotics-Academy.

Programming robots is not an easy task and it is advisable to avoid exposing the students to the whole complexity immediately. Therefore, the application for each exercise is divided into a specific template code and the student's code. The template contains a code that resolves several auxiliary aspects like Graphical User Interface (GUI) or connections with drivers, leaving the core of the algorithm to be programmed by the student. There are two different templates developed in Robotics-Academy: Python ROS nodes and Python Jupyter Notebooks.

3.1. Physical and Simulated Robots Availability: Gazebo

One of the first design decisions was not to focus the exercises on a specific robot, but to choose a variety of platforms: drones, robot TurtleBot with wheels, cars, humanoids, etc. with the idea of being able to design practical sessions that cover different aspects of robotics without the limitation of a particular robot typology or model. This contrasts with other teaching suites focused on a specific robot platform like [17].

Figure 2 shows some of the robots supported by the current version of the teaching suite. These include indoor robots with wheels and differential traction, such as the TurtleBot robot (real and simulated), or outdoor ones like a Formula-1 car and a simulated taxi. Drones typology is also supported, for instance, Parrot ArDrone is included in both real and simulated versions. It also offers the possibility of practical activities with sensors such as cameras, lasers, Global Positioning System (GPS), RGB-D cameras (Kinect, Xtion), etc.

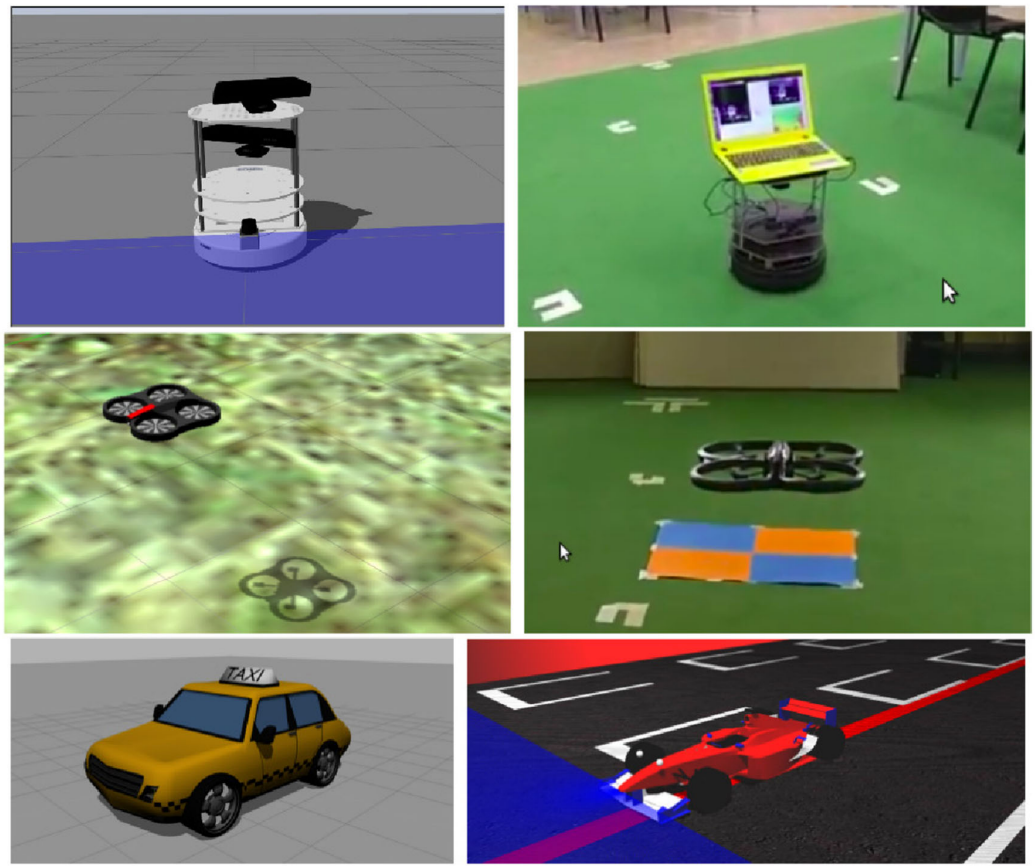

Figure 2. Some of the robots supported in Robotics-Academy.

Simulators are frequently used in robotics, both in research and in teaching. Some of their advantages in education are:

- Simulators allow robotics to be learnt with complex and powerful robots without the real hardware. They facilitate learning for many students, as sessions can be conducted with more learners than real available robots. They are ideal, for example, for mass online courses.
- In a simulator the programming failures by students do not damage any real hardware.
- It is possible to reduce the complexity of some problems taking information from the simulator. For example, the simulator provides the true absolute position of the robot at any time. The student may take it for granted and focus only on the control code, not the self-localization.
- They allow fair comparisons for gamification, and results indicate that games and/or simulations have a positive impact on learning goals [41]. With simulators, the code of all students can be executed exactly on the same machine and in the same simulated world.
- They enable development of automatic evaluators for the assessment of the student's solutions. These evaluators can be connected to the simulated world collecting all the data needed to evaluate the quality of the student's solution.

Experience with real platforms is academically very valuable in itself, although it has some practical drawbacks too. The software design for Robotics-Academy allows a balance between practical sessions with simulated robots and with physical robots (when available). It permits the same academic application to be connected to a physical robot or its simulated counterpart with only a few configuration changes.

There are many simulators available: V-REP, Webots, etc. All the practical sessions presented in this paper for teaching intelligent robotics were designed to work with the Gazebo simulator [42], which has been chosen as a reference (http://gazebosim.org). It is free software, of high quality,

maintained by the Open Source Robotics Foundation (OSRF) and has great acceptance in the international robotics community. The version currently used in Robotics-Academy is Gazebo-9.

3.2. Drivers and Middleware: Robot Operating System (ROS)

In recent years, many environments have appeared (*middleware*), such as ROS, Open Robot Control Software (Orocos) and Yet Another Robot Platform (YARP), that simplify programming of applications for robots. Middleware is very useful for shortening the development time and increasing the robustness of robotics applications, because well-tested software pieces are reused.

ROS [20] was chosen as reference middleware for Robotics-Academy. It is the de facto standard and has a larger user and developer community than any other middleware. It provides an extensive set of drivers, including drivers for robotic arms, indoor robots, drones, etc.

It was initially implemented in C++ but now supports many programming languages (including Python). It was created for Linux machines but now tends to be multi-platform (recently Windows support was announced). Currently, the version used in Robotics-Academy is ROS Melodic.

3.3. Exercise Templates as Python ROS Nodes

One of the templates used in Robotics-Academy for exercises are ROS Python nodes, as shown in Figure 3. They contain two parts: an already programmed part and the student's code.

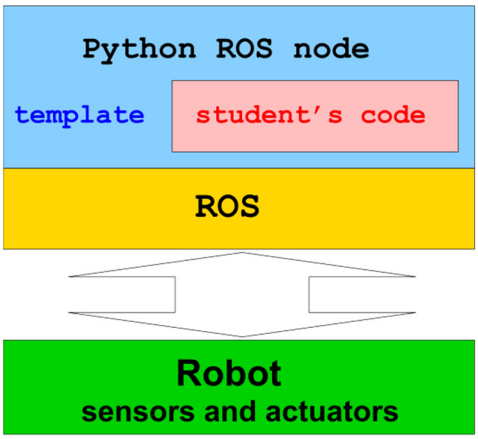

Figure 3. Design of a robotics exercise with a Python ROS node.

The already programmed part acts as a container where the student code is incorporated and provides several contributions:

1. It solves the GUI using PyQt (https://pypi.org/project/PyQt5/), necessary for debugging, and provides a simple Application Programming Interface (GUI API) for the student to use which is ad hoc and academically illustrative for that specific exercise.
2. It provides a simple, local programming interface for sensors and actuators (Hardware Abstraction Layer API).
3. It provides a timing skeleton for robot behaviors.

The GUI allows sensory data or even partial processing suitable for each exercise to be shown. For example, the student can include code to show if a color filter is working well on the image of the robot's camera, or even show the path selected by the planning algorithm in the simulated world of the robot. As all this functionality is already provided, the student is not distracted by programming graphical functionalities, but can benefit from them to debug. PyQt is used for this part.

The Python ROS node conceals the complexity and details of the robotic *middleware* used. It connects to the sensors and actuators of the robot using the appropriate ROS methods and communication mechanisms (topics, subscriptions, publishing, …), but provides a simple local API

in Python for the student. In this way, the student simply invokes Python methods to update the data collected by the sensors and to send commands to the actuators. ROSpy is used for this part.

The most common timing skeleton for robot behaviors is a continuous loop of iterations. Each iteration executes four steps: collecting sensory data, processing them, deciding actions and sending orders to the actuators. This timing engine is very suitable for materializing reactive behaviors. The Python ROS nodes in Robotics-Academy provide a Callback function that must be filled by the student and is invoked from the timing engine typically 15 times per second.

The typical architecture of an exercise with a simulator is shown in Figure 4. The Gazebo simulator materializes the behavior of the physical robot in a virtual scenario. The drivers are plugins that run inside the simulator itself and provide access to the sensors and actuators of the simulated robots. In order to start a world simulation for a specific exercise, a configuration file that determines the scenario, the robots involved, etc. is created and used. Gazebo provides a local viewer to observe the evolution of the simulated world, including the behavior of the robot programmed by the student and also allowing interaction with the scene by adding certain element at runtime, stopping, re-launching the simulation, etc. The Python ROS node is also executed through a configuration file.

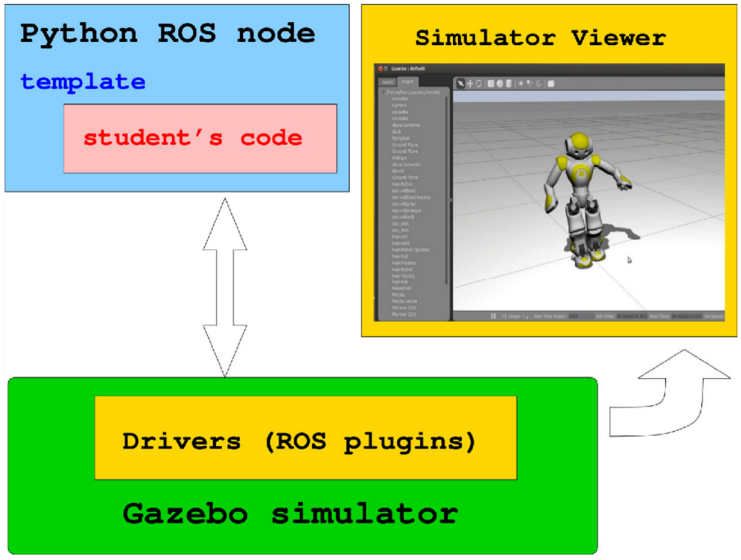

Figure 4. Typical execution of an exercise of Robotics-Academy using Gazebo viewer.

3.4. Exercise Templates as Jupyter Notebooks

The second template used in Robotics-Academy for exercises is Jupyter Notebooks (http://jupyter.org/) [43,44]. They also provide GUI-API, HAL-API and a timing skeleton for robot behaviors to the student, but inside a web browser.

The web-based Notebooks support rich documents that combine code and computational results with text narratives, mathematics, images, video and any media that a modern browser can display. They are a highly useful tool for teaching, presenting, and sharing computational work. Their new architecture allows users to work in any language, with implementations in Python, R, Julia and Haskell.

In Robotics-Academy, a Notebook has been created for each exercise following the design shown in Figure 5. This includes code cells where the student may write Python code. It then connects with an underlying Python interactive shell (IPython, renamed Jupyter kernel) by a specific protocol named Zero Message Queue. The interactive shell already executes the Python code and connects to the ROS drivers of the robot using ROSpy library.

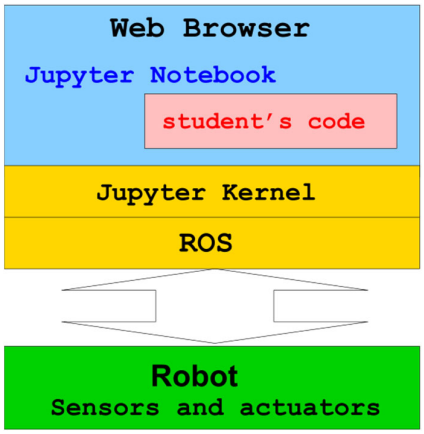

Figure 5. Design of a robotics exercise with Jupyter Notebook in the web browser.

This new document format provides several educational advantages. First, theory can also be included in the Notebooks, which helps to explain the related theoretical concepts to the student, helping them to successfully solve the corresponding exercise. Second, graphics can also be included in the Notebooks, which may show the result of any computation on the student's code or be used for debugging. Third, Notebooks can be shared. Fourth, no additional editor program is needed, the web browser is directly used for writing the code, and browsers are multi-platform.

There are also some drawbacks when using Jupyter Notebooks for robotics exercises. The GUI is much slower than Python ROS nodes. For instance, real time image display (at 30 fps) is not possible. Running through the web browser and the IPython kernel adds a computing overhead which prevents real-time applications.

In order to execute the student's solution for an exercise, the Jupyter Notebook is run from the browser, connecting to the simulated robot (Gazebo simulator) using the Jupyter Kernel (Figure 6).

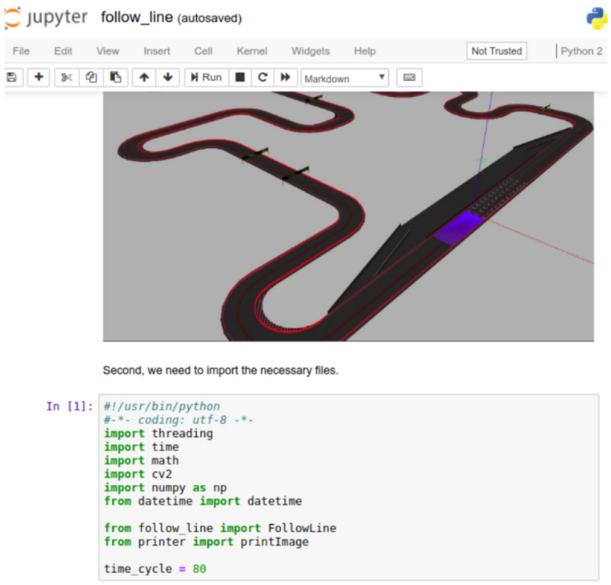

Figure 6. A Notebook example for Formula-1 follow-line exercise, including code cells and the simulated world.

3.5. Installation and Distribution

The presented framework is open source, available at GitHub (https://github.com/JdeRobot/RoboticsAcademy). It is open to collaborations, extensions and

modifications. Teachers may create their own exercises and submit them to the public repository of the platform.

Easy installation of the framework is very important, so the student does not get lost in the middleware and may start programming the robotics exercises from the very first day. Special care has been taken in Robotics-Academy to make it very simple to install and use. All its underlying infrastructure (Python, Gazebo, ROS, Jupyter) is provided as regular and standard Debian packages, and so they can be installed easily. Both ROS middleware and Gazebo are well maintained by OSRF for Ubuntu Linux, which releases official Debian packages for them. There are also official Jupyter packages and instructions for their installation. In addition, the required underlying resources which are not contained in standard Gazebo or ROS packages are created and stored in several Debian packages from the JdeRobot organization.

The created Python ROS nodes and Jupyter Notebooks for the exercises are provided as source code which needs to be cloned on the student's machine from the Robotics-Academy GitHub repository. The whole set is presented as a collection of independent exercises. Each exercise contains its specific templates and the corresponding configuration files. Designing them as self-contained makes this collection easily extensible.

Robotics-Academy is multi-platform. It natively runs on Ubuntu Linux computers, and Docker images have been created and are maintained for Windows and MacOS users, so they can have access to the framework on a virtual machine (Docker container) in their computer.

3.6. Available Exercises

The collection of available exercises is accessible on the web (https://jderobot.github.io/RoboticsAcademy), and deals with different problems of service robotics, control, automation and computer vision. Attractive practical problems were chosen, aiming to develop solutions for service robots that have recently emerged in society: autonomous cars, robotic vacuum cleaners, drones, etc. In this way, the student is involved in real-life applications, motivated and will find easier to understand the direct usefulness of the concepts.

Currently there are 18 exercises already available, as shown in Table 2, and the collection is expandable. Any subset of exercises can be extracted at will to support the practical side of any robotics course.

Table 2. List of exercises available at Robotics-Academy.

	Mobile Robots
MR1	Bump and go behavior
MR2	Roomba cleaning an apartment (without self-location)
MR3	Roomba cleaning an apartment (with self-location)
MR4	Follow line behavior
	Autonomous Cars
AC1	Formula1: follow a line in a race circuit
AC2	Local navigation with Virtual Force Field (VFF)
AC3	TeleTaxi: Global navigation with Gradient Path Planning (GPP)
AC4	TeleTaxi: Global navigation with Open Motion Planning Library (OMPL)
AC5	Autonomous car negotiating a crossroad
AC6	Autonomous car: parallel parking
	Drones
D1	Navigation by position
D2	Drone following an object on the ground
D3	Drone following a road with visual control
D4	Drone cat and mouse
D5	Escape from a maze following visual clues
D6	Searching for victims within a perimeter
	Computer Vision
CV1	Color filter
CV2	3D reconstruction from a stereo pair

4. Intelligent Robotics Course

An introductory course to Intelligent Robotics has been developed, using Robotics-Academy as a platform for practical exercises. The syllabus of the proposed course covers these units:

- Unit 1. Introduction: Robot products and applications (vacuum cleaners, autonomous driving, factories, etc.), software for robotics.
- Unit 2. Sensors and actuators: Laser, light detection and ranging (LIDAR), cameras, Pulse Width Modulation (PWM) motors, servos, locomotion systems, manipulation systems.
- Unit 3. Reactive systems: Control loop, proportional-integral-derivative (PID) controllers, Finite State Machines, behavior-based robotics.
- Unit 4. Map building: Occupancy grids, topological maps.
- Unit 5. Robot Navigation: Local navigation algorithms (VFF, CVM), path planning algorithms for global navigation (A *, GPP, visibility graphs).
- Unit 6. Self-localization for robots: Probabilistic techniques, particle filters, visual SLAM.

This course follows the 'learn by doing' paradigm, focusing on the programming of robotics algorithms. Its practical contents are publicly available and ready to use. Its most representative exercises are detailed below, describing the robot used, the scenario and the task to be solved. Reference solutions have been also developed for all exercises, they are used to record and publish videos so students may see in advance the expected robot behavior on each exercise. In addition, automatic evaluators have been programmed, providing automatic feedback to the students and encouraging them to improve their solutions.

4.1. Bump and Go Behavior

The challenge of this exercise (MR1) is to program a simple bump and go behavior for an indoor robot like the TurtleBot2. It has to wander advancing in a straight line until it meets an obstacle, then go back for a while, and randomly turn to avoid the obstacle.

The TurtleBot2 robot has two driving wheels, with differential traction, to which commands are made through the intermediate interface of movement: forward speed V and speed of rotation W. It is equipped with odometry and a depth sensor (RPLidar in the real TurtleBot, Hokuyo laser in the simulated one).

This behavior can be solved with three simple states: "forward", "backward" and "turn" as can be seen in the upper part of Figure 7. This exercise shows the power of finite state machines (FSM) when programming robot intelligence. The student learns the use of states and transitions, switching for example from "forward" to "backward" when the laser sensor detects an obstacle that is too close, or switching from "turning" to "forward" after a random time interval. This exercise was also tested with a real TurtleBot2 robot as shown in the bottom part of Figure 7.

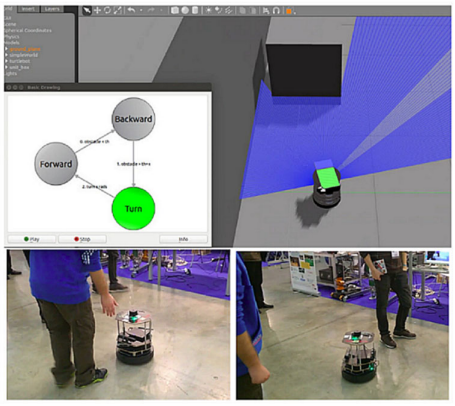

Figure 7. Simulated and real Turtlebot with a finite state machine (FSM) for wandering without hitting any obstacle.

4.2. Drone Cat and Mouse

This exercise (D4) proposes a game of cat and mouse between two drones. The student is asked to develop a behavior for the drone that acts like the cat to chase and stay close to the mouse drone. In the simulated world the mouse drone is red and moves autonomously. Figure 8 shows the world in Gazebo, free of obstacles (only far trees), with the mouse (red drone) and the cat (black drone). For this exercise, a set of mice of several difficulty levels were prepared. The simplest is slow and predictable in its movement. The most difficult one is the fastest and most unpredictable in its movement.

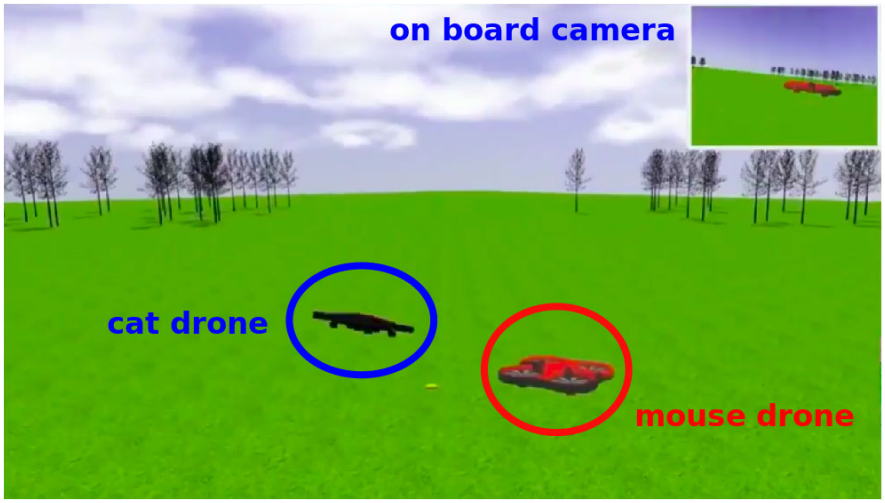

Figure 8. Gazebo scenario for "Drone cat and mouse" exercise.

The drone is equipped with two cameras, front and vertical, inclinometers and GPS. It has four rotors operated with an intermediate global interface that accepts commands to move the whole drone in 3 axes: forward/backward, elevation/descent, lateral displacements and rotations. This aerial robot is similar to Parrot's ArDrone2. The local API provides simple methods in Python to access these sensors and movement interface. These include the method **sensor.getImage()** to obtain images from the camera, **pose.getPose3D()** to get the drone's 3D position from two inclinometers and GPS (as an XYZ vector and a quaternion for orientation), the methods **extra.takeoff()** and **extra.land()** to take off and land, and the method **cmdvel.sendCMDVel()** to send movement commands to the drone.

This exercise was designed to teach basic reactive control in robotics, including PID controllers, as well as introducing the students into basic image processing, for example color filters, morphological filters or the segmentation of the other drone in an image. The typical solution performs a switch-case based control, including the automatic search if the mouse is lost and the regular case of vision-based monitoring, where a PID reactive control works successfully accelerating when the mouse becomes small, rising when the mouse is in the upper part of the image, etc.

A code snippet is shown in Figure 9, containing an example of student's code. The code template calls the **execute()** method iteratively ten times per second, and the student fills it with its own solution code (inside the orange rectangle). It includes code for sensor reading, image processing, the PID control computation and the motor command on each iteration.

```
cat.drone.takeoff()
def execute(self):
    global error_alt
    global derivative_alt
    global integral_alt

    red_lower=(0,5,0)
    red_upper=(8,255,255)

    #SENSOR READING
    droneImage = self.getImage()

    #IMAGE PROCESSING
    image_hsv = cv2.cvtColor(droneImage, cv2.COLOR_RGB2HSV)
    image_mask = cv2.inRange(image_hsv, red_lower,red_upper)
    cnts = cv2.findContours(image_mask,
    cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
    if len(cnts) > 0:
        for cnt in cnts:
            x_cnt,y_cnt,w_cnt,h_cnt = cv2.boundingRect(cnt)
            ...

            # PID CONTROL
            kp_alt =0.05
            kd_alt = 0.1
            ki_alt = 0
            new_error_alt = 120 - (y+(h/2))
            proportional_alt = kp_alt * new_error_alt
            rate_of_change_alt = new_error_alt -error_alt
            error_alt = new_error_alt
            derivative_alt = kd_alt * rate_of_change_alt
            integral_alt = integral_alt + error_alt
            integral_alt = ki_alt * integral_alt
            output_alt = proportional_alt + derivative_alt + integral_alt

            # MOTOR COMMAND
            self.drone.sendCMDVel(output_dist,0,0,0,0,output_side)
```

Figure 9. Code snippet showing part of the exercise solution (student´s code inside the orange rectangle).

This exercise includes an evaluator which automatically measures the quality of the student's solution. It continuously computes the spatial distance between the two drones and computes the number of seconds below a certain threshold. Figure 10 shows the graphic interface of this automatic evaluator and the evolution of that spatial distance over time in an example.

The test consists of several rounds of two minutes with the successive mice. The score is the number of seconds the cat-drone code developed by the student is close enough to the mouse-drone (distance is less than 3 m).

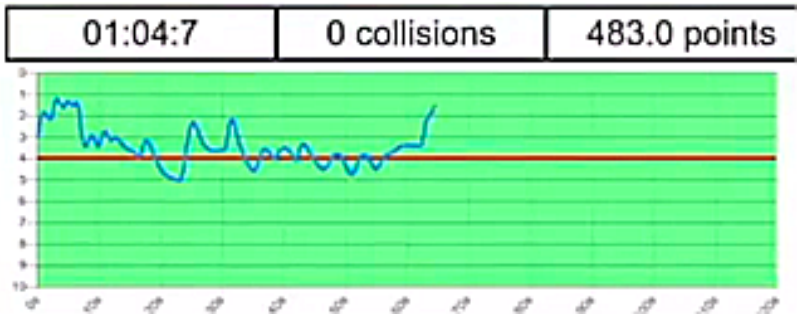

Figure 10. Automatic evaluator for "Drone cat and mouse" exercise.

4.3. Roomba Cleaning an Apartment

The proposed problem in this exercise (MR3) is to program a high-end Roomba vacuum cleaner to clean an apartment. The robot is intended to autonomously visit the whole area of the apartment,

the faster the better. Revisiting the same area twice or more does not increase the area already cleaned. It can softly bump into obstacles like walls, chairs, etc., as real vacuum cleaners do.

The robot has motors which can be commanded in translation speed V and rotation speed W. It includes a laser sensor (the purple area in Figure 11) and encoders as main sensors. The local API offers simple methods in Python to access these sensors and actuators. For instance, the **getLaserData()** method is used to read the laser measurements. The vacuum cleaner also has an onboard camera which is used for estimating the 3D position of the robot. Its position estimations are available using the **pose.getPose3D()** method.

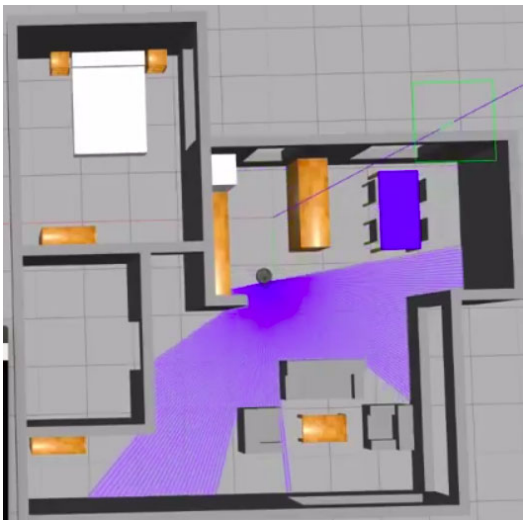

Figure 11. Gazebo scenario for "Roomba cleaning an apartment" exercise.

To solve this task, random wandering algorithms may be developed. For example, initial spiral and bump and go behavior may be proposed as base solutions. However, this exercise has been designed to teach some planning algorithms. As self-localization is provided, more efficient foraging algorithms such as [45,46] are also available choices to be implemented in order to solve this exercise. They minimize revisiting places and sweep the apartment more systematically and efficiently. This exercise includes an evaluator which automatically measures the quality of the student's solution. As shown in Figure 12, it displays the path followed, measures the area cleaned and shows the remaining time of the test. The final score is the percentage of apartment cleaned after a 10-min round.

Figure 12. Automatic evaluator for "Roomba cleaning an apartment" exercise.

4.4. Autonomous Car: Parallel Parking

The challenge of this exercise (AC6) is to program an autonomous car (sedan like a Tesla) to perform parallel parking in a street. The scenario provided in the Gazebo simulator is shown in Figure 13, with the street almost full of green cars, but with a gap big enough for the yellow autonomous car.

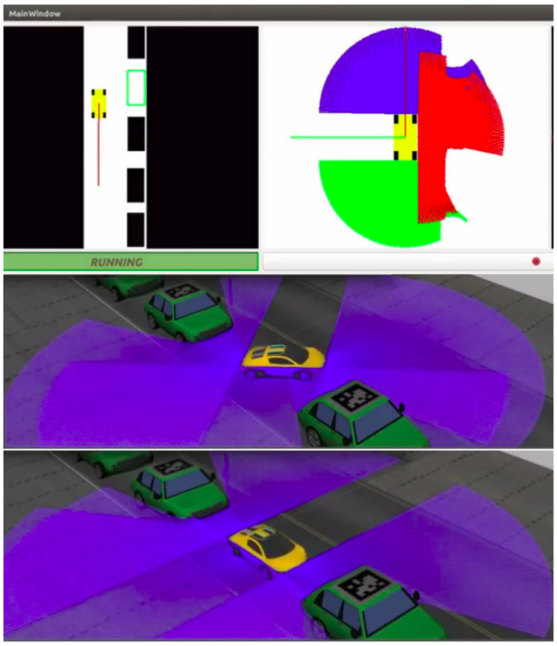

Figure 13. Autopark exercise on autonomous car.

The robot is a yellow sedan equipped with odometry (encoders) and three laser sensors on board (blue for front, red for right side, light green for back). Regarding movement, it has a steering wheel and two driving wheels, but supports an intermediate command interface: forward speed V and speed of rotation W. The robot has inertia and does not instantly achieve the speeds commanded by the software. This car model was created for the Gazebo simulator, with a realistic appearance and embedded sensors.

Several algorithm choices such as [47–49] are available to be implemented by students. For instance, it has a fully reactive approach with case-based control and a definition of several cases according to the distances from the front car, the back car and side empty space. A second reference solution performed by students includes a path planner and a position-based pilot that follows the path. The OMPL and its path planning for non-holonomic robots were used.

The automatic evaluator measures the time taken to complete the parking maneuver, the final orientation of the car (approximately parallel to the street) and the distances to front and rear cars already parked.

5. Experimental Results

The proposed teaching suite was used on a bachelor's degree course in Telematics (Robotics subject) and a master's degree course in computer vision at Rey Juan Carlos University (Vision in Robotics subject). It was also used in 6 introductory courses (up to 20 h) to robotics and drone programming. More than 150 students used it. This provides first experimental validation and preliminary feedback from real users.

In all these courses 50% of the time was dedicated to theory, while the other 50% was to practical individual exercises (both online and on-site). The exam included an oral presentation for each exercise where each student showed and explained their solution answering questions about the code they

developed. In addition, the students wrote blogs describing their projects including text, images and videos. These blogs helped them to share their insights and promote improvement and competition.

The master's degree students completed two exercises: (AC1) follow line and (CV2) 3D reconstruction from a stereo pair. The undergraduate students accomplished five exercises: (MR1) bump and go behavior, (AC1) follow line, (D4) drone cat and mouse, (AC2) Formula-1 local navigation with VFF and (AC4) TeleTaxi: Global navigation with GPP. The students with the best results under simulation were given the opportunity to test and improve their algorithms using the real TurtleBot2 robots available at the laboratory.

The surveys completed by the students showed a favorable reaction to the educational tool and its exercises, with 95% of evaluations being equal to or higher than 5 points out of 10, and 80% of evaluations being equal to or higher than 8 points, as shown in Table 3.

For instance, two representative open comments from students were: "I like this tool because it allows me to develop several algorithms without worrying about the required infrastructure" and "With Robotics-Academy you may program robots easily, even without previous experience in robotics". They also revealed that the evaluation of the installation is positive, although it should be further simplified.

Table 3. Information from students' surveys.

Global Evaluation of Robotics-Academy (0 = Extremely Bad; 10 = Extremely Good)											
Score	0	1	2	3	4	5	6	7	8	9	10
Percentage	0	0	0	2.5	2.5	2.5	12.8	28.2	20.6	25.7	5.2
Difficulty of Installation of Robotics-Academy (0 = Extremely Difficult; 10 = Extremely Easy)											
Score	0	1	2	3	4	5	6	7	8	9	10
Percentage	2.6	0	2.6	5.1	2.6	20.5	15.4	15.4	17.8	15.4	2.6

In 2019, Robotics-Academy was used in a pilot study with 221 pre-university students (167 who were17 years old and 54 who were 18 years old). Five exercises with their Jupyter Notebook templates were used to introduce those students to Python and robot programming (https://www.rtve.es/alacarta/videos/la-aventura-del-saber/aventura-del-saber-pensamiento-computacional-bachillerato/5273296/). A control group and an experimental group took the same "Technology, Programming and Robotics" course, the former group in a traditional way and the latter using Robotics-Academy. Quantitative analyses of the impact of the tool were performed [50] using the Computational Thinking Test [51] and six different Bebras problems [52]. The experimental and objective results showed that Robotics-Academy had a small positive impact for 17-year-old students and a more significant positive impact on 18-year-olds. For both ages, the learning experience is better using Robotics-Academy.

In addition, the "Drone cat and mouse" exercise was successfully used in the three editions of the robot programming competition PROGRAM-A-ROBOT (https://jderobot.github.io/activities/competitions/2018). The last one was celebrated at the IROS-2018 conference (https://www.youtube.com/watch?v=VFBL6zuXqgo). The automatic evaluator was very useful to compute and for seeing the participant's score in real time.

In the courses, the exercises were presented as competitive games (gamification), which makes them more appealing and fun. In our experience of teaching and organizing robotic competitions this gamification increases a student's motivation and results. Thus, several exercises are presented as a type of competition: which drone spends more time close to another one? How long does it take a programmed Formula-1 car to complete a lap following a line? etc. The developed evaluators are useful for providing an instant and objective score in the simulator.

The open source Robotics-Academy repository is currently active, having more than 90 forks (repository change proposals) in the last months.

6. Conclusions

A new open educational resource, named Robotics-Academy, has been presented for teaching robotics in higher education. An introductory Intelligent Robotics course has been presented. It is based on Robotics-Academy, publicly available and ready to use. The course includes several appealing exercises related to drones, autonomous cars, mobile robots and computer vision. Both the Intelligent Robotics course and the underlying Robotics-Academy platform are the main contributions of this article.

The proposed open educational resource was preliminarily validated in several engineering courses (Bachelor's and Master's) at the Rey Juan Carlos University. Over 150 students used it and more than 95% of them gave a positive evaluation of the framework in the surveys. It has also been validated with 221 pre-university students. An objective quantitative analysis performed shows that the Robotics-Academy tool had a positive impact on students' learning results, when compared to the control group.

Robotics-Academy integrates five remarkable features. Their combination makes the platform unique, advantageous and different from other available programs for robotics education presented in the related work section.

First, the programming language should not be too complex as it is not the learning focus, only a tool for learning robotic concepts. The Python language was selected for its simplicity and expressive power.

Second, Robotics-Academy supports both simulated and physical robots. Simulators provide many advantages when teaching robotics, although the experience with real robots is certainly valuable as well. The Gazebo simulator was chosen as reference because it is open source, a standard in the community, and developed by experts. As it supports many different robot types, it widens the range of implementable exercises, providing a greater range than educational frameworks which focus on one type of robot only, as SyRoTek, Tekkotsu or Robots Formation Control Platform.

Third, ROS has recently been incorporated as the underlying middleware for Robotics-Academy. Major code refactoring and rewriting have been performed to adapt the whole platform. This makes our open educational resource more general, easier to install, easier to maintain and widens the number of potential users, as ROS is the de facto standard in robot programming.

Fourth, the robotic software systems are complex, and it is advisable to expose students gradually to such complexity, especially beginners. To conceal this complexity a template was programmed for each robotic exercise. Each template provides a simple interface (HAL-API) to access the robot sensors and actuators, wrapping and hiding the middleware details. The template also solves many auxiliary tasks such as the specific graphical interface (GUI-API) for each exercise and its timing engine. Those auxiliary tasks are required to have a running robot application but have little educational value for an introductory course. Two types of template have been created in Robotics-Academy: Python ROS nodes and Jupyter Notebooks. The students have to simply complete these templates adding there the code of their solutions.

Fifth, installation of the framework is easy as it is mainly undertaken through Debian packages. Developing the code templates was a time-consuming task, but it is an essential contribution of Robotics-Academy. Templates make a difference for the students by letting them focus on the algorithms and providing simple, pleasing and useful programming interfaces. The complexity of the middleware and of the Graphical User Interface is hidden from students. This contrasts with other approaches, such as Robot Ignite Academy, which exposes the students to all ROS messages and inner workings.

The programming of several automatic evaluators is another important contribution. They provide instant feedback for students and allow gamified educative approaches [53]. The students' performance and motivation are bound to improve using them. To our knowledge, no other available platform for robotics education provides such automatic evaluators.

With respect to the first original release of Robotics-Academy [38], the programming language has been changed from C++ to Python, the support for several robot types has been integrated (instead of only the Pioneer 2DX robot), and the platform provides a specific template for each

exercise (instead of a generic template for all) allowing more illustrative GUIs. Regarding the upgrades from the previous release [39], new exercises have been developed, ROS (instead of the previously used ICE) has been fully integrated as underlying robotics middleware and Jupyter Notebooks have been added as an alternative template for the exercises. In addition, automatic assessment has been incorporated as a new feature.

In order to improve Robotics-Academy, several lines are under development:

- new exercises are being added such as one with a robot arm, a Formula-1 race with several autonomous cars competing at the same time, and another in the Amazon logistics context.
- we are upgrading the educative framework to work with ROS2 as underlying infrastructure.
- we are working on providing this educational resource as a pure web online application, so students around the world can use it on their computers without any installation at all.
- a comprehensive experimental analysis of the student's experience when using this platform is planned, creating a complete questionnaire and collecting information from more students. This improvement will surely provide more solid methodological bases for the proposed platform.

Author Contributions: Conceptualization, J.M.C., E.P., L.G.-P. and J.F.-C.; Funding acquisition, J.M.C.; Investigation, J.M.C., E.P., L.G.-P. and J.F.-C.; Methodology, E.P.; Project administration, J.M.C.; Resources, J.M.C.; Software, J.M.C., E.P., L.G.-P. and J.F.-C.; Supervision, J.M.C.; Validation, L.G.-P. and J.F.-C.; Visualization, L.G.-P. and J.F.-C.; Writing—original draft, E.P.; Writing—review and editing, L.G.-P. and J.F.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially funded by the Community of Madrid through the RoboCity2030-DIH-CM (S2018/NMT-4331) and by the Spanish Ministry of Economy and Competitiveness through the RETOGAR project (TIN2016-76515-R). The authors thank Google for funding the JdeRobot non-profit organization in its calls for Google Summer of Code 2015, 2017, 2018, 2019 and 2020.

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

References

1. Touretzky, D. Robotics for computer scientists: what's the big idea? *Comput. Sci. Educ.* **2013**, *23*, 349–367.
2. Aliane, N. Teaching fundamentals of robotics to computer scientists. *Comput. Appl. Eng. Educ.* **2011**, *19*, 615–620.
3. Berenguel, M.; Rodriguez, R.; Moreno, J.; Guzman, J.; Gonzalez, R. Tools and methodologies for teaching robotics in computer science & engineering studies. *Comput. Appl. Eng. Educ.* **2016**, *24*, 202–214.
4. Mateo, T.; Andujar, J. Simulation tool for teaching and learning 3d kinematics workspaces of serial robotic arms with up to 5-dof. *Comput. Appl. Eng. Educ.* **2012**, *20*, 750–761.
5. Mateo, T.; Andujar, J. 3D-RAS: A new educational simulation tool for kinematics analysis of anthropomorphic robotic arms. *Int. J. Eng. Educ.* **2011**, *27*, 225–237.
6. Lopez-Nicolas, G.; Romeo, A.; Guerrero, J. Active learning in robotics based on simulation tools. *Comput. Appl. Eng. Educ.* **2014**, *22*, 509–515.
7. Jara, C.A.; Candelas, F.A.; Puente, S.; Torres, F. Hands-on experiences of undergraduate students in automatics and robotics using a virtual and remote laboratory. *Comput. Educ.* **2011**, *57*, 2451–2461.
8. Jara, C.; Candelas, F.; Pomares, J.; Torres, F. Java software platform for the development of advanced robotic virtual laboratories. *Comput. Appl. Eng. Educ.* **2013**, *21*, 14–30.
9. Fabregas, E.; Farias, G.; Dormido-Canto, S.; Guinaldo, M.; Sanchez, J.; Dormido, S. Platform for teaching mobile robotics. *J. Intell. Robot Syst.* **2016**, *81*, 131–143.
10. Kulich, M.; Chudoba, J.; Kosnar, K.; Krajnik, T.; Faigl, J.; Preucil, L. Syrotek-distance teaching of mobile robotics. *IEEE Trans. Educ.* **2013**, *56*, 18–23.
11. Esposito, J.M. The state of robotics education: Proposed goals for positively transforming robotics education at postsecondary institutions. *IEEE Robot. Autom. Mag.* **2017**, *24*, 157–164.
12. Gil, A.; Reinoso, O.; Marin, J.; Paya, L.; Ruiz, J. Development and deployment of a new robotics toolbox for education. *Comput. Appl. Eng. Educ.* **2015**, *23*, 443–454.
13. Gonzalez, R.; Mahulea, C.; Kloetzer, M. A matlab-based interactive simulator for mobile robotics. In Proceedings of the 2015 IEEE International Conference on Automation Science and Engineering (CASE), Gothenburg, Sweden, 24–28 August 2015.

14. Corke, P. Robotics, Toolbox for Matlab (Release 9). 2015. Available online: http://www.petercorke.com/ (accessed on 15 September 2016)
15. Corke, P. *Robotics, Vision and Control: Fundamental Algorithms*; Springer Tracts in Advanced Robotics; Springer: Berlin/Heidelberg, Germany, 2011.
16. Guzman, J.; Berenguel, M.; Rodriguez, F.; Dormido, S. An interactive tool for mobile robot motion planning. *Robot. Auton. Syst.* **2008**, *56*, 396–409.
17. Guyot, L.; Heiniger, N.; Michel, O.; Rohrer, F. Teaching robotics with an open curriculum based on the e-puck robot, simulations and competitions. In *Proceedings of 2nd International Conference on Robotics in Education (RiE 2011) Vienna, Austria, 15–16 September 2011;* Stelzer, R., Jafarmadar, K., Eds.; Ghent University: Ghent, Belgium, 2011, pp. 53–58.
18. Farias, G.; Fabregas, E.; Peralta, E.; Vargas, H.; Dormido-Canto, S.; Dormido, S. Development of an easy-to-use multi-agent platform for teaching mobile robotics. *IEEE Access* **2019**, *7*, 55885–55897.
19. Detry, R.; Corke, P.; Freese, M. TRS: An Open-Source Recipe for Teaching/Learning Robotics with a Simulator. 2014. Available online: http://ulgrobotics.github.io/trs (accessed on 15 September 2016.).
20. Quigley, M.; Gerkey, B.; Smart, W.D. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*; O'Reilly Media, Inc.: Newton, MS, USA, 2015.
21. Martinez, A.; Fernandez, E. *Learning ROS for Robotics Programming*; Packt Publishing Ltd.: Birmingham, UK, 2013.
22. Mahtani, A.; Sanchez, L.; Fernandez, E.; Martinez, A. *Effective Robotics Programming with ROS.*; Packt Publishing Ltd.: Birmingham, UK, 2016.
23. Joseph, L. *Mastering ROS for Robotics Programming*; Packt Publishing Ltd.: Birmingham, UK, 2015.
24. Cooney, M.; Yang, C.; Arunesh, S.; Padi Siva, A.; David, J. Teaching robotics with robot operating system (ros): A behavior model perspective. In *Workshop on "Teaching Robotics with ROS"*; European Robotics Forum: Tampere, Finland, 2018.
25. Alisher, K.; Alexander, K.; Alexandr, B. Control of the mobile robots with ros in robotics courses. *Procedia Eng.* **2015**, *100*, 1475–1484.
26. Michieletto, S.; Ghidoni, S.; Pagello, E.; Moro, M.; Menegatti, E. Why teach robotics using ROS? *J. Autom. Mob. Robot. Intell. Syst.* **2014**, *8*, 60–68.
27. Ferreira, N.F.; Araújo, A.; Couceiro, M.; Portugal, D. Intensive summer course in robotics – Robotcraft. *Appl. Comput. Inform.* **2020**, doi:10.1016/j.aci.2018.04.005.
28. Hold-Geoffroy, Y.; Gardner, M.A.; Gagne, C.; Latulippe, M.; Giguere, P. ros4mat: A matlab programming interface for remote operations of ros-based robotic devices in an educational context. In Proceedings of the International Conference on Computer and Robot Vision (CRV), Regina, SK, Canada, 28–31 May 2013; pp. 242–248.
29. Avanzato, R.L.; Wilcox, C.G. Work in progress: Introductory mobile robotics and computer vision laboratories using ros and matlab. In Proceedings of the ASEE Annual Conference and Exposition, Conference Proceedings, Salt Lake City, UT, USA, 23–27 June 2018.
30. Ruiz-Sarmiento, J.R.; Galindo, C.; Gonzalez-Jimenez, J. Experiences on a motivational learning approach for robotics in undergraduate courses. In Proceedings of the 11th International Technology, Education and Development Conference, INTED 2017, Valencia, Spain, 6–8 March 2017; pp. 3803–38011.
31. Ortega-Arranz, A.; Sanz-Martinez, L.; Alvarez-Alvarez, S.; Munoz-Cristobal, J.A.; Bote-Lorenzo, M.L.; Martinez-Mones, A.; Dimitriadis, Y. From low-scale to collaborative, gamified and massive-scale courses: Redesigning a mooc. In *European Conference on Massive Open Online Courses*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 77–87.
32. Corke, P.; Greener, E.; Philip, R. An Innovative Educational Change: Massive Open Online Courses in Robotics and Robotic Vision. *IEEE Robot. Autom. Mag.* **2016**, *23*, 1.
33. Pozzi, M.; Malvezzi, M.; Prattichizzo, D. Mooc on the art of grasping and manipulation in robotics: Design choices and lessons learned. In *International Conference on Robotics and Education RiE 2017, Sofia, Bulgaria, 26–28 April*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 71–78.
34. Tellez, R.; Ezquerro, A.; Rodriguez, M.A. *ROS in 5 Days: Entirely Practical Robot Operating System Training*; Independently Published: Madrid, Spain, 2016.
35. Casañ, G.A.; Cervera, E.; Moughlbay, A.A.; Alemany, J.; Martinet, P. ROS-based online robot programming for remote education and training. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 6101–6106.

36. Cervera, E.; Martinet, P.; Marin, R.; Moughlbay, A.A.; Del Pobil, A.P.; Alemany, J.R.; Casañ, G. The robot programming network. *J. Intell. Robot. Syst.* **2016**, *81*, 77–95.
37. Casañ, G.; Cervera, E. The Experience of the Robot Programming Network Initiative. *J. Robot.* **2018**, *2018*, 1–15.
38. Cañas, J.; Martin, L.J. Innovating in robotics education with gazebo simulator and jderobot framework. In Proceedings of the XXII Congreso Universitario de Innovación Educativa en Enseñanzas Técnicas CUIEET, Alcoy, Spain, 17–19 June 2014; pp. 1483–1496.
39. Cañas, J.M.; Martin, A.; Perdices, E.; Rivas, F.; Calvo, R. Entorno docente universitario para la programación de los robots. *Rev. Iberoam. Automática E Inf. Ind.* **2018**, *15*, 404–415.
40. Joseph, L. *Learning Robotics Using Python*; Packt Publishing: Birmingham, UK, 2015.
41. Vlachopoulos, D.; Makri, A. The effect of games and simulations on higher education: A systematic literature review. *Int. J. Educ. Technol. High. Educ.* **2017**, *14*, 22.
42. Koenig, N.P.; Howard, A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 4, Sendai, Japan, 28 September–2 October 2004; pp. 2149–2154.
43. Ragan-Kelley, M.; Perez, F.; Granger, B.; Kluyver, T.; Ivanov, P.; Frederic, J.; Bussonnier, M. The jupyter/ipython architecture: A unified view of computational research, from interactive exploration to communication and publication. In *AGU Fall Meeting Abstracts*; AGU: Washington, USA, 2014.
44. Kluyver, T.; Ragan-Kelley, B.; Perez, F.; Granger, B.E.; Bussonier, M.; Frederic, J.; Kelley, K.; Hamrick, J.; Grout, J.; Corlay, S.; et al. Jupyter Notebooks—A publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*; Loizides, F., Schmidt, B., Eds.; IOS Press: Amsterdam, The Netherlands, 2016; pp. 87–90.
45. Ntawumenyikizaba, A.; Viet, H.H.; Chung, T. An Online Complete Coverage Algorithm for Cleaning Robots based on Boustrophedon Motions and A* search. In Proceedings of the 8th International Conference on Information Science and Digital Content Technology (ICIDT2012), Jeju, Korea, 26–28 June 2012.
46. Gonzalez, E.; Alvarez, O.; Diaz, Y.; Parra, C.; Bustacara, C. BSA: A complete coverage algorithm. In Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 18–22 April 2005; pp. 2040–2044.
47. Zips, P.; Böck, M.; Kugi, A. Optimisation based path planning for car parking in narrow environments. *Robot. Auton. Syst.* **2016**, *79*, 1–11.
48. Hsieh, M.F.; Ozguner, U. A parking algorithm for an autonomous vehicle. In Proceedings of the 2008 IEEE Intelligent Vehicles Symposium, Eindhoven, The Netherlands, 4–6 June 2008; pp. 1155–1160.
49. Kim, D.; Chung, W.; Park, S. Practical motion planning for car-parking control in narrow environment. *IET Control Theory Appl.* **2010**, *4*, 129–139.
50. Instituto Nacional de Tecnologías Educativas y de Formación del Profesorado (INTEF). La Escuela de Pensamiento Computacional y su Impacto en el Aprendizaje. 2018–2019. 2019. Available online: https://intef.es/wp-content/uploads/2019/12/Impacto_EscueladePensamientoComputacional_Curso2018-2019.pdf (accessed 25 September 2020).
51. Román-González, M.; Pérez-González, J.C.; Moreno-León, J.; Robles, G. Can computational talent be detected? Predictive validity of the Computational Thinking Test. *Int. J. Child Comput. Interact.* **2018**, *18*, 47–58.
52. Dagiene, V.; Futschek, G. Bebras international contest on informatics and computer literacy: Criteria for good tasks. In *International Conference on Informatics in Secondary Schools, Evolution and Perspectives*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 19–30.
53. Dichev, C.; Dicheva, D. Gamifying education: What is known, what is believed and what remains uncertain: A critical review. *Int. J. Educ. Technol. High. Educ.* **2017**, *14*, 9.