Why teach robotics using ROS

Stefano Michieletto, Stefano Ghidoni, Enrico Pagello, Michele Moro, and Emanuele Menegatti
Intelligent Autonomous Systems Lab (IAS-Lab)
Department of Information Engineering (DEI)
Faculty of Engineering, The University of Padua
Via Ognissanti 72, I-35129 Padua, Italy
{stefano.michieletto, stefano.ghidoni, enrico.pagello, michele.moro, emanuele.menegatti}@dei.unipd.it

***Abstract*—This paper focuses on the key role played by the adoption of a framework in teaching robotics with a computer science approach in the master in Computer Engineering. The framework adopted is the Robot Operating System (ROS), which is becoming a standard *de facto* inside the robotics community. The educational activities proposed in this paper are based on a constructionist approach. The Mindstorms NXT robot kit is adopted to trigger the learning challenge. The ROS framework is exploited to drive the students programming methodology during the laboratory activities and to allow students to exercise with the major computer programming paradigms and the best programming practices. The major robotics topics students are involved with are: acquiring data from sensors, connecting sensors to the robot, and navigate the robot to reach the final goal. The positive effects given by this approach are highlighted in this paper by comparing the work recently produced by students with the work produced in the previous years in which ROS was not yet adopted and many different software tools and languages were used. The results of a questionnaire are reported showing that we achieved the didactical objectives we expected as instructors.**

***Index Terms*—Educational robotics, ROS, LEGO NXT robot, teaching robotics.**

I. Introduction

Robotics is a multidisciplinary field which involves researchers from many different research areas, from mechanics to control theory, from electronics to computer science. Thus, robotic competences are taught at different levels and from different points of view through undergraduate and graduate courses. In this paper, we focus on a course on *Autonomous Robotics* offered in the master curriculum of Computer Engineering at the University of Padua.

In our case this is the only course strictly related to Robotics: the lack of previous experience, apart from a general basic knowledge, makes it not easy for the students to tackle the complexity which is behind the building of autonomous robots. As reported also by [20], incremental experiences are essential for this purpose but, even providing a well-planned sequence of experiences, we realized that the code developed by students suffered of one of the major plague of robotics: i.e. little (or almost no) code reuse. Even though reusing code is difficult and possibly involves debugging and integration effort, it is nevertheless an important aspect of software development that should be learned. The absence of code reuse in the old course implementation, was caused by the fact that we used different software tools and programming languages (and this is often the case for robotics courses, see for instance [23], [24]). This is a situation similar to what happens in the robotics community. However, the solution comes from software environments able to offer to different programs the possibility to communicate one with each other sharing a common interface: in a few words, a software framework.

The choice of exploiting a software framework offers several advantages in a high number of real-world robotics applications, and therefore in such scenarios it is often a compulsory choice. But when educational purposes are concerned, some further motivations should justify its adoption: the advantages provided by a framework cannot be fully exploited in this scenario, since time usually allocated to laboratorial activities is short, and the robotic platform exploited is often quite simple. A very common robot for teaching activities is the Mindstorms NXT: this option is relatively cheap and simple and for these reasons rather frequently adopted in university courses [12][7][4]. Moreover, it should be noted that the NXT platform comes with the NXC language that targets all the specific capabilities of the robot: this means we are comparing a small, easy to learn but hard to spend, procedural programming language targeted to the robotics platform employed for the experiments, with a large, general and complex framework with high potential which is however tricky to understand, for which the NXT is just one of the many platform that can be handled.

The purpose of this paper is to show that integrating a general purpose framework into a university course has a positive didactic impact. Students were able to use a framework to complete the same experiments developed using NXC in the previous years, and the chosen framework (ROS) showed good performance also when used to handle a simple robotic platform. The success of this choice relies on exploiting the framework for the laboratorial experiences without substantial overhead for the students. This way students develop their knowledge in robotics by using tools that are easier to apply to real-world scenarios in which they will be asked to work after graduating.

Given the decision of employing a framework for teaching robotics, an important aspect is to choose which one is best suited, since in the last decade a lot of robotics frameworks have been developed. This is the case of URBI [1], OROCOS [3], YARP [6], Microsoft Robotics Studio [8] and Piaget [9]; however, no one has obtained the proper consensus to become a standard *de facto*. Recently, the scene has changed thanks to the introduction of the Robot Operating System (ROS) [13]. ROS is a framework for robotics with the addition of some operating system functionalities. A great variety of tools are integrated in order to allow easy debug operations

and analyze the communication between processes. One of the main advantages that ROS offers among other similar products is the large community that supports, uses and extends the software.

The choice of employing ROS for teaching robotics is important to let the students have experience of a complete and modern software framework for robotics. Moreover, since ROS is likely to become the most popular choice in the future, supporting an increasing number of robotic platforms, its knowledge will enable students to easily handle other robots in the future. Many universities are adopting ROS to teach robotics, including South Carolina, Washington, Brown, Stanford, KU Leuven, Sherbrooke, Tokyo, Sapienza and Leibniz University.

The paper is organized as follows: in section II the robotic course organization will be described, together with the expertise that students should gain with it; in section III the experience with laboratory activities will be summarized, and a comparison between before and after the introduction of ROS will be provided. In section IV the didactic impact of the lab experiences will be evaluated, based on the analysis of the homeworks produced by students, as well as on their answers to a questionnaire. Finally, in section V some final remarks on the choice of employing ROS for teaching robotics will be drawn.

II. Robotics Course for Master in Computer Engineering

The robotics course is based on a mixed approach merging theoretical lectures in class and practical experiences in the laboratory. Lectures aim at building a strong background on robotics fundamentals, perception systems, computer vision, and navigation, while laboratory sessions are meant to let students get acquainted with software tools and algorithms exploited in robotics.

The platform we chose is the Mindstorms NXT 2.0. Several robot kits are available for educational purposes [25], but we believe the NXT offers the right balance of complexity versus modularity [17] (in Italian). NXT is composed by a microcomputer, three motors, and a suite of sensors, including touch sensors, sonars, light sensors, microphones, compass and accelerometers. A set of LEGO parts also comes in the box, letting the user build structures for holding sensors in the preferred position, as shown in figure 1: in (a), a sketch of the model employed in the laboratory experience is shown, equipped with a light sensor (blue bubble) and a sonar (red bubble); in (b), a robot holding an omnidirectional camera.

The strong point of this package is that the LEGO kit provides high flexibility, so that it is possible to build robots with a wide variety of shapes, and choose among a number of different sensor placements. The basic configuration shown in figure 1 (a) (usually called “Tribot”) is one of the simplest to control, and has therefore been selected for first laboratory experiences; it is a differential drive robot platform with two driven wheels in the front and a castor wheel on the back. Motion control is rather simple in this case, but some complexity is added by the fact that motors themselves are not very precise, and the way they are controlled (e.g. acceleration curves) has an impact on the trajectories the robot can follow.

(a) (b)

Fig. 1. The NXT in the Tribot configuration (a) with the light sensor (blue) and the ultrasonic sensor (red). In (b), a robot holding an omnidirectional camera placed on a green carpet is shown.

III. Robotic Experiences

In this chapter the teaching experience before and after the introduction of a software framework will be outlined.

A. Past experiences

In the previous years, we taught students the open-source C-like programming language NXC (Not eXactly C) and we used the IDE called BricxCC. With this approach, programs are composed of one or more tasks, and the syntax is easy to read and understand for people with little programming background.

However, it should be noted that the NXC is dedicated to program the NXT platform and even if it is very popular, its knowledge is not exploitable outside this context. Moreover, the NXC language is limited to the sensors included in the NXT package, and it is hard to deal with additional sensors, like a webcam attached to the robot. In such cases some additional software and libraries should be run outside NXC to manage the new sensors.

Robotic frameworks are going towards module-based architectures, often supporting distributed processing, and capable of exploiting network resources for sharing data. Of course, exploitation of such aspects is beyond the scope of the robotics course, however, by employing such new frameworks even for developing the first laboratory experiences with simple robots, it is possible to introduce concepts that will be reused by students when they will face more complex scenarios. Adopting ROS instead of relying on simpler languages, as NXC, presents of course an overhead [10] and a steeper learning curve, but from our experience it was clear that the overhead was limited: the number of hours dedicated to laboratory sessions was the same in the course adopting ROS as in the one in which NXC was employed.

B. Recent experience

Updating the Autonomous Robotics MSc course, a single framework has been adopted in order to make students familiar with programming practices which could be useful in a future job.

As previously discussed, a number of frameworks for robotics exist, some of which are specifically tailored for

didactic purposes, as it is the case of Tekkotsu [18], that cares about real-time aspects of robot programming, and Pyro (Python Robotics) [19]. Almost all of such frameworks are able to support the basic experiences that are proposed during the course, so the main aspects considered choosing the framework to be adopted were: i) the possibility to exploit the same framework also outside the context of the course, i.e., the generality of the framework; ii) the number of supported robots suited for lab experiences (in particular the NXT); iii) the community and the documentation, that represent a valuable help for students. So for example, by adopting Tekkotsu there is a strong constraint on the types of robots that are supported, probably caused by the fact that it is a very recent project. The ROS framework is instead very strong on this point, and has recently become even stronger thanks to the ROS industrial project [5], that is meant to improve the support for industrial robots, making it a proper working tool for professional engineers in the robotics field.

The effectiveness of ROS in teaching is demonstrated by the rather large number of robotics course that have adopted it, including Brown University (USA), Cornell University (USA), University of Birmingham (UK) and obviously Stanford University (USA). The panorama is quite wide, since the robots employed among the courses are quite different, and the tasks assigned to students depend on this: for example, experiences with inverse kinematics are proposed with the PR2 robotic platform. Anyway, a common base about motion planning and basic computer vision can be found in the majority of the courses.

The introduction of ROS did not require to change the laboratory experiences objectives developed in previous years. Such experiences focus on the quantitative aspects typical of engineering and to create a constructivism/constructionism and educational robotics architecture [11]. All didactical goals of the course were kept. In addition, we could add other objectives to the course related to the computer science curriculum: students have the opportunity to write code in a widely used programming language (preferably C++, commonly used in the robotics field) supporting Object-Oriented Programming (OOP) concepts. Among all available frameworks, ROS has been chosen since it supports OOP, and also because its community is very active, and represents a valuable help. A large variety of tutorials are available from which students can easily learn.

In the following, the set of laboratory experiences proposed in the course will be described. They involve the classic challenges of robotics: robot control, navigation, and perception through sensory information. This way students can gain experience on different aspects, and build a robot that has a certain degree of intelligence. Multiple sensors have been employed: ultrasonic proximity sensor, color sensor, and an omnidirectional camera: this way it is possible to increase the complexity of the sensing tasks in the different laboratory experiences.

Experience 1: Obstacle avoidance

In the first experience, students have to plan a robot path in order to avoid two obstacles, represented by cones. The robot has to go around the first cone and stop 3 cm behind the second one for 10 seconds, and finally come back, as shown in Figure 2.

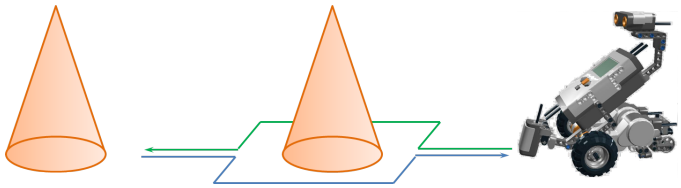


Fig. 2. Robot behavior for the first experience

Robotics objectives: The first objective is to make students familiar with robots, and their motion and sensors. They have to deal with noisy sonar sensor, motor inaccuracy and odometry imprecision. During this experience, students work with basic ROS modules: they start practicing with ROS master (`roscore` or `.launch` files), then they explore nodes and topics functionalities (`rostopic` and `rxgraph`). Once students are familiar with these basic concepts, they can evaluate robot characteristics by teleoperating it using the keyboard. A simple visualizer (`rviz`) is also available, which eases the result visualization. In order to do this we developed a basic package to provide students the NXT model, the robot controller to translate velocity command into joint rotations and the teleoperation program. Finally, students create a new package and develop their own module, which communicate with the others following the publisher/subscriber mechanism, which is exploited also for reading sensors, e.g. acquiring range data, and for controlling robot movements. The experience involves robotics topics like interpreting uncertain sensor data, navigation and control, and motion planning.

Computer science objectives: The experience is meant to make students review the concepts of data structure and class. They can understand how data are handled in the framework by looking at prebuilt ROS messages, and they are also asked to analyze the package structure in order to know how they depend one from each other; this way they will be able to develop their own packages in a similar way. Students will also analyze the callback construct which is covered through the publisher/subscriber communication method. In this first experience a simple problem is proposed, so that students can try to solve it in a fast procedural way as they usually do. Nevertheless, an object oriented approach is promoted to build independent entities to control the robot.

Experience 2: Path planning

The goal of the second experience is robot navigation into a map composed of *N*x*M* cells, that are represented using a green carpet with a white grid, over which the robot moves. Students have to extend the algorithm created in the first experience to make the robot go from a *Start* cell to a *Goal* cell while avoiding obstacles, and understanding its own motion inside the map. To guide the robot motion a color sensor is exploited in order to identify the squares borders: as long as the robot moves, the sensor indicates whether it is going over the green carpet or a white stripe. The representation of the robot movements is simplified, since it can only go to the north, south, east, west squares, that is, it cannot move diagonally; this means the activities for navigating across the map are

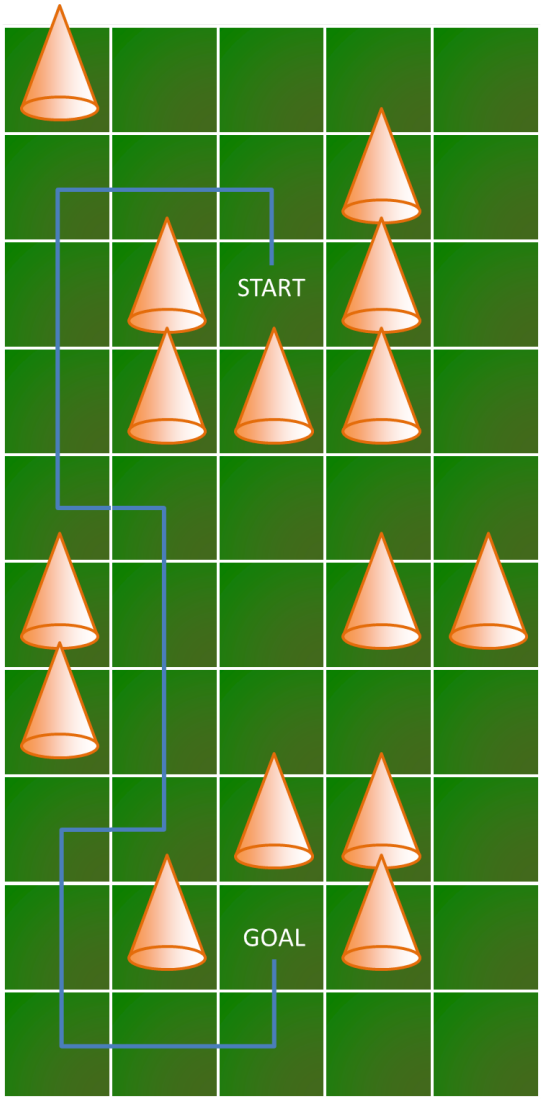


Fig. 3. Map example

basically forward translations by one cell and rotations. Since employed motors are not extremely precise, the orientation of the robot is affected by inaccuracies that students need to deal with: once the robot crosses a white stripe, it performs left and right rotations in order to understand its direction with respect to the map grid.

The experience is divided into two scenarios: in the first one, obstacles in the map are fixed and students know where they are, whereas in the second one they can be moved, hence the correct route cannot be known in advance. Figure 3 shows an example of map and a possible path to reach the goal.

Once the robot has reached the goal, it is asked to push a touch sensor, which is in turn connected with ROS. This tells the system the experiment was successful.

Robotics objectives: In this experience students have to develop a path planning algorithm choosing among those presented in the course. They deal with the intensity sensor data and exploit communication through ROS modules. Understanding modules is very important working with ROS. Students are pushed to split their code into a reasonable set of modules that should communicate among each other. Using sensory information the robot has to recognize the motion from one cell to another, navigate in the map and avoid obstacles in order to reach the *Goal*. Since the map is not known in the second scenario, an internal representation of the checked cells and robot position should be developed. The purpose of the last part of the experience is to learn how to exploit communication between two different devices (the robot and the touch sensor) using ROS. The main robotics topics faced by students in this experience are localizing and mapping, and multiple-robot coordination.

Computer science objectives: A more complex experience highlights the importance of a well-structured code. Students need to model the robot and the map as two different entities, and to generate a message flow that starts from the former and goes into the latter. The change from a static map to a dynamic one also helps students to reconsider their choices in software design in order to better identify which data belongs to which structure. Finally, the communication between the robot moving on the map and the one handling the touch sensor promote the abstraction of the robot class. The first one should implement a very smart behavior in order to move and manage several sensors, while the second one has to take care of the touch sensor only. Both elements are the same entity (a robot), so they comes from the same generic class, but each one is a different specific class.

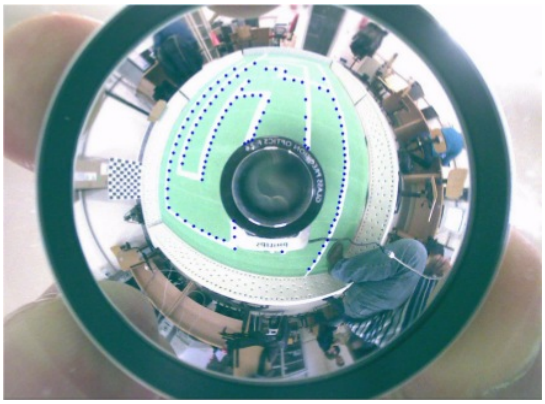
Experience 3: Perception using computer vision

In this lab experience a new sensor, based on computer vision, is introduced. The robot is given the same task of *Experience 2*, that is, reaching a goal on a map with moving obstacles, but this time the sensory suite is different: in place of the infrared sensor, an omnidirectional camera is exploited.

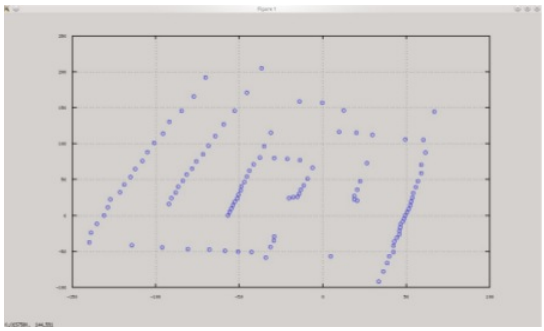
As a first step, students are asked to build their own omnidirectional sensor and calibrate it with the Ocam Calib tool [14]. They should then find the optimal placing on the robot, which involves two main parameters: the height from the ground, and the orientation of the axis of the mirror, which can be vertical or slightly tilted. At this point the real laboratory experience, divided into two parts, starts.

As a first task, students are asked to implement a vision-based module that behaves like the infrared sensor of the previous experience, thus providing the same output. This can be achieved by selecting a small Region Of Interest (ROI) in the image, in which objects at a certain distance will appear, and observing its content.

The second task is more focused on robotic vision, since students are asked to develop a ROS module capable of analyzing all white stripes visible on the carpet. In particular, the lines which are closer should be detected to evaluate their distances to the robot, in order to gather its position inside the cell and its orientation with respect to the grid; in figure 4 it is possible to see the result when the white stripes composing a labyrinth are analyzed. The resulting module is able to overcome the data provided by the infrared sensor: thanks to the vision sensor, it is possible to detect stripes at a rather high distance, and to recover the orientation of the robot with



(a) Lego NXT starting configuration.



(b) Lego NXT omnidirectional configuration.

Fig. 4. Result of the line detection algorithms when a labyrinth is analyzed. (a), and a bird's eye view reconstructed from such image (b).

respect to the grid by looking at a single image, instead of performing rotations while passing over a line.

Robotics objectives: In previous experiences students have already developed some hands-on robots knowledge, therefore guidelines do not specify details on how to place the vision sensor, nor how to develop vision algorithms. Students are asked to explore different solutions, and find the best-suited ones: for example, by properly tilting the omnidirectional camera it is possible to extend the detection range, but this leads to a slightly more complex processing. Facing the trade-off between complexity and accuracy is an important aspect at this stage. Another important objective is to face computer vision with omnidirectional images, which is widely used in robotics. Students will learn how to build an omnidirectional sensor, calibrate it, and exploit the huge amount of information included in each single image. Finally, experience with ROS modules and how to manage communication among them will be developed: students will create a module for acquiring images, and use its output to feed another module implementing image processing algorithms. This experience combines the image acquisition and processing topics together with localizing and mapping topics inherited from the preceding experience.

Computer science objectives: One of main goals of this experience is the creation of a module that can replace an existing sensor module and then improve it with new features. This implies code modularity, and also encourage the use of inheritance and design patterns in order to make the work easier when classes are well-designed. Using an environment similar to the one introduced in the second experience helps students to focus on the computer vision activity; however, students that chose a good software design in the previous experience will be able to easily reuse it in this third one.

IV. Evaluation of didactic impact

In order to obtain a feedback on the effectiveness of the adoption of ROS, we compared the results of the group of students that attended the Autonomous Robotics course two years ago before we introduced ROS and the group of last year, that exploited the ROS framework. The comparison exploits on one hand some objective considerations about problem-solving methods and code developing and on the other hand the subjective students' opinions about laboratory experiences, satisfaction about the course, and future expectations.

A. Comparison with the past

The use of ROS as a common framework pushes students to develop algorithms in a structured environment. This was not the case in the previous years, since NXC is still a procedural programming language. Developing code in NXC limits students to work in a procedural paradigm. NXC provides functions or data structures, but students do not actually use them, because they feel they can code faster if they write their algorithms in a single file. The analysis of the code of last year reveled students favored cut-and-paste to think to a general structure for the code; for the same reason, they preferred to use global variables to passing parameters, and created ad-hoc solutions to problem generalization. While using ROS, this year's students are pushed to organize their software into modules, reuse their data structures and classes, exploit class inheritance. They will also experience the power of message sharing mechanism, which is an important service offered by robotics frameworks.

The proposed experiences are designed so that students can have an advantage if they are able to correctly formalize the problem and choose a proper software design for implementing solutions, since in this way each experience can be built on top of the previous ones. In Table I the results of the analysis of the software produced by this year's students is reported. The source code (about 1000 lines for each experience) of sixteen students (grouped by two) has been analyzed looking for the following OOP concepts:

- features coded in functions or methods to be applied in different situations (**code reuse**);
- similar characteristics and values related to each other in order to build a single structure (**structured data**);
- modeling of real entities into objects that represents them and their features (**classes**).

Data reported in Table I represent the percentage of students groups implementing each OOP concept at least once in their source code. As it can be seen, code reuse and adoption of structured data strongly increased after the first experience.

B. Students satisfaction

Students were asked to fill an anonymous questionnaire summarized in Table II. The answer to each question is

TABLE I. DATA FROM COURSE STUDENTS AFTER THE INTRODUCTION OF A COMMON FRAMEWORK.

	1^{st} **exp.**	2^{nd} **exp.**	3^{rd} **exp.**
Code reuse	75%	100%	100%
Structured data	38%	88%	100%
Classes	63%	88%	88%

represented by a choice among four states: *Very much* (green), *Enough* (blue), *A little* (red), *Not at all* (yellow). Some of the statements are quite similar, in order to emphasize small differences that could be hidden by a multiple choice answer.

The questionnaire was meant to test several aspects of the laboratory activity, like:

- exploitation of students' background in terms of programming skills and software engineering;
- effort spent in developing the experiences;
- closeness with future job activities.

Answers to the questionnaire highlight that programming capabilities students were sufficient, even though they would like to had more programming experience (Q2); moreover, such capabilities improved during the practical experience in the laboratory (Q3). From the software engineering point of view, it turned out that the ROS framework forced students to adopt a modular approach for their software, which eased its reuse (Q4). Students appreciated team work, and are convinced that it is very difficult to achieve important results by working alone (Q7-Q8). Students showed a moderate confidence on the fact that expertise coming from lab experiences could be reused in a future job (Q10,Q16): answers to these questions were based on the current working experience that a certain number of students already had, while the others answered based on what already graduated colleagues told them. Students seemed to greatly appreciate the hands-on approach of the course, and would agree on increasing the number of courses adopting this approach (Q9, Q13), even though this means an increase in work load (Q11-Q12). Finally, students also appreciated the way experiences gradually increase in complexity (Q14-Q15).

Overall, the questionnaire demonstrates that choices made while designing the course had a good impact over several aspects, including code production, putting into practice theoretical concepts studied during lectures, and working with a structured framework like ROS.

V. CONCLUSION

This paper presented a series of experiences targeted to MSc students attending the Autonomous Robotics course. Experiences already defined for the course in the previous years, based on the constructivist approach, are now developed inside a robotic framework, that forces students to get in touch with advanced software structure, and take advantage of the services it offers. The introduction of ROS as a framework pushes students to use OOP concepts thanks to the highly structured environment they have to work with. The overhead given by understanding and learning how to use a new framework, besides its intrinsic added value, is compensated by the later ease to develop code for the subsequent laboratory experiences, to integrate new sensors, and to interact with different devices. Finally, being able to handle complex softwares like ROS is a strong reward for students, which realize they have learnt how to deal with real (and complex) robotic frameworks.

The course includes a set of laboratory experiences that represent an important feedback for students' achievements. Experiences test the capability of controlling a robot (experience 1), sensing the environment with simple sensors and modifying the robot's behavior accordingly (experience 2) and handling more complex sensors that need high-level processing (experience 3). This can be seen as a small but complete set of abilities students should gain to deal with robots, and the positive outcome of such experiences is the ultimate proof of their achievements. For this reason, the way experiences cover a number of subjects, and their increasing complexity level has been stressed in the paper.

The analysis of a report for each laboratory experience and of the developed code made it possible to verify students' comprehension of robotics basics, their use of complex syntactic constructs and their problem-solving capabilities. Finally, students' satisfaction was tested by means of a questionnaire. The results highlight a good response both regarding how students' expectations were met, as well as improvements in robotics and programming skills. This has also been assessed by testing the robots moving on the map, and observing how they deal with a dynamic environment. Since all students were able to correctly complete all the experiences, even though going through a number of difficulties, it is possible to conclude that the proposed set of experiments is correctly planned.

The laboratory experiences are currently limited by both the robots used (more advanced ones would require stronger investments) and by the time that can be dedicated to experiences. If such limitations could be overcome, an interesting extension would be to face challenges posed by humanoid robots, starting from the gait. More advanced tasks, like grasping, are too complex to be practically solved in the context of this course. The introduction of the approach presented in this paper into other courses than Autonomous Robotics is also planned.

REFERENCES

[1] J. Baillie. URBI: Towards a Universal Robotic Low-Level Programming Language. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3219–3224. IEEE, 2005.

[2] J. C. Baillie. Urbi 2: Introduction to concurrent real-time programming. In *Proceedings of the Third International Workshop on Software Development and Integration in Robotics*. IEEE ICRA, May 2008.

[3] H. Bruyninckx, P. Soetens, and B. Koninckx. The real-time motion control core of the Orocos project. In *IEEE International Conference on Robotics and Automation*, pages 2766–2771, 2003.

[4] A. Cruz-Martin. J.A. Fernandez-Madrigal, C. Galindo, J. Gonzalez-Jimenez, C. Stockmans-Daou, and J.L. Blanco-Claraco. A LEGO Mindstorms NXT approach for teaching at Data Acquisition, Control Systems Engineering and Real-Time Systems undergraduate courses. *Computers & Education*, 59(3):974–988, Nov. 2012.

[5] S. Edwards, and C. Lewis. Ros-industrial:applying the robot operating system (ros) to industrial applications. *IEEE International Conference on Robotics and Automation*, ECHORD Workshop, St Paul, Minnesota, May 2012.

[6] P. Fitzpatrick, G. Metta, and L. Natale. Towards long-lived robot genes. *Robot. Auton. Syst.*, 56(1):29–45, Jan. 2008.

[7] J.M. Gomez-de-Gabriel, A. Mandow, J. Fernandez-Lozano, and A.J. Garcia-Cerezo. Using LEGO NXT Mobile Robots With LabVIEW for Undergraduate Courses on Mechatronics. *IEEE Trans. on Education*, 54(1):41–47, Feb. 2011.

[8] J. Jackson. Microsoft robotics studio: A technical introduction. *IEEE Robotics & Automation Magazine*, 14(4):82–87, Dec. 2007.

[9] J.D. Dessimoz, P.F. Gauthey, and H. Omori, Piaget Environment for the Development and Intelligent Control of Mobile, Cooperative Agents and Industrial Robots. *International Symposium for Robotics (ISR)*, Aug. 2012.

[10] T. Koletschka, and A. Hofmann. Technikum Wien's entry in the Robotour'11 competition. In *Proceedings of the 2nd International Conference on Robotics in Education (Rie2011)*, pages 151–156, Wien, Austria, September 2011.

[11] E. Menegatti, and M. Moro. Educational robotics from high-school to master of science. In *SIMPAR Workshop on Teaching Robotics, Teaching with Robotics*, pages 484–493, Darmstadt, Germany, November 2010.

[12] P. Ranganathan, R. Schultz, and M. Mardani. Use Of Lego NXT Mindstorms Brick In Engineering Education. In *Proceedings of the 2008 ASEE North Midwest Section Conference Educating the Engineer of 2020*, Platteville, Wisconsin, USA, October 2008.

[13] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[14] D. Scaramuzza, A. Martinelli, and R. Siegwart. A toolbox for easy calibrating omnidirectional cameras. In *Proc. of The IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[15] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots, 2001. In *Artificial Intelligence*, Volume 128, Issues 1?2, May 2001, Pages 99-141.

[16] J. Arlegui, E. Menegatti, M. Moro, A. Pina. Robotics, Computer Science curricula and Interdisciplinary activities In *Proceedings of the TERECoP Workshop "Teaching with robotics: didactic approaches and experiences* pp.10-21 November 3-4, 2008 Venice (Italy).

[17] M. Moro, E. Menegatti, F. Sella, M. Perona. Imparare con la Robotica - Applicazioni di problem solving. Edizioni Erickson Trento (Italy), pp.1-196, 2011.

[18] E. Tira-Thompson, D.S. Touretzky. The Tekkotsu robotics development environment. In *Proc. IEEE ICRA 2011* pp. 6084-6089 May, 2011.

[19] D. Blank, D. Kumar, L. Meeden, H. Yanco. Pyro: A python-based versatile programming environment for teaching robotics. In *Journal on Educational Resources in Computing (JERIC)*, Volume 4, Number 3, p.3, 2004.

[20] A. Nourdine. Teaching fundamentals of robotics to computer scientists. In *Computer Applications in Engineering Education* Volume 19, Issue 3, pages 615–620, September 2011.

[21] Robot Operating System. *www.ros.org*

[22] Terecop project: Teacher Education on Robotics-Enhanced Constructivist Pedagogical Methods. *www.terecop.eu*

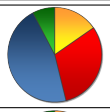
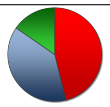
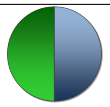
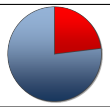
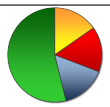
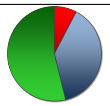
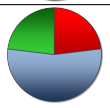
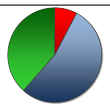
[23] G. Cielniak, N. Bellotto, T. Duckett Integrating Vision and Robotics into the Computer Science Curriculum In *Workshop on Teaching Robotics, Teaching with Robotics*, pages 484–493, Riva del Garda, Italy, April 2012.

[24] A. Albers, M. Frietsch, V. Bartenbach, G. Robens, N. Burkhardt. A New Robotics Laboratory for Interdisciplinary Mechatronic Education In *SIMPAR Workshop on Teaching Robotics, Teaching with Robotics*, pages 456–464, Darmstadt, Germany, November 2010.

[25] M. Ruzzenente, M. Koo, K. Nielsen, L. Grespan, and P. Fiorini, A review of robotics kits for tertiary education In *Workshop on Teaching Robotics, Teaching with Robotics*, pages 44–50, Riva del Garda, Italy, April 2012.

TABLE II. RESULTS OF THE QUESTIONNAIRE.

Legend: Not at all A little Enough Very much

Q1.	During the experiences of the robotics course I exploited knowledge acquired in other courses that I had not put into practice.	
Q2.	Programming skills I developed in previous courses were sufficient for working on the ROS framework.	
Q3.	I gained new programming experience because I had to work with a complex framework as ROS is.	
Q4.	In order to efficiently work with ROS, I have been forced to divide my software into modules, which in turn made it easier to reuse it.	
Q5.	In my future job I will be asked to work with modular software structures similar to ROS.	
Q6.	By working with ROS I have got in touch with a well structured software, that has been a source of inspiration for new ideas on how to develop my software.	
Q7.	By working in groups I improved my ability to split tasks among people.	
Q8.	By working in groups we were able to reach results I would not have reached alone.	
Q9.	By working with real robots we needed to deal with practical problems that would have not shown up in a simulated environment.	
Q10.	Thanks to the lab experiences I developed expertise I will exploit in my future job.	
Q11.	Lab experiences require a lot of time to be completed, but are the only way I can develop expertise that are useful for my future job.	
Q12.	Lab experiences required an excessive work load, and should correspond to a higher number of credits.	
Q13.	I would like that a larger number of courses would be based on a mixed approach including both theory and laboratory experiences, in order to put into practice what we learn during lectures.	
Q14.	The complexity of lab experiences increases gradually.	
Q15.	What I learnt in each experience has been useful for the following ones.	
Q16.	In my future job, I will be asked to work on topics that are similar to the ones faced in the laboratory.