Seamless Robot Simulation Integration for Education: A Case Study

Wolfgang Hönig[1], Arash Tavakoli[2], and Nora Ayanian[1]

***Abstract*— We present a seamless integration of a robotics simulator and hardware implementation, in which the same code can be executed unchanged in simulation or on a physical robot. We discuss the design challenges and our resulting architecture, which seamlessly integrates V-REP, a widely used robotics simulator, with iRobot Create 2 robots. Our approach is easy to set up, simple to understand, yet powerful enough to be useful beyond the classroom. We demonstrate the versatility of our approach by describing an undergraduate-level course in robotics that uses our framework, comparing it to a previous course which did not use simulation tools.**

I. INTRODUCTION

Simulation tools play an important role in robotics in academia and industry because they enable early testing of algorithms without the risk of damaging robots or infrastructure, or the risk of harming humans in the physical world. However, simulations may require a significant time commitment as they often rely on specialized software. For example, GAZEBO [1], a widely used simulator, currently only works with the Ubuntu operating system and is very powerful when used in combination with the *Robot Operating System* (ROS) [2]. Furthermore, simulation often executes software that is different from the one running on the actual robot. This can result in issues caused by software bugs which may arise during hardware experiments and not in simulation, and vice versa.

In an educational setting, the two aforementioned issues often cause course designers to concentrate either on simulation verification of algorithms or testing on physical hardware, but rarely on both. Good practice in academia and industry is to test algorithms in various settings by iterating them in simulation first, then testing them on physical platforms. Therefore, it would be ideal to practice this approach in classes. To the best of our knowledge, most courses which have attempted to seamlessly combine simulation and physical verification have used simulation platforms and robots which are not industry- or research-standard—e.g., LEGO Mindstorms robot kit [3]—and are often highly limited in capabilities and extensibility.

This paper presents an approach that seamlessly integrates simulation and physical execution. In particular, a user can write Python code which can be executed without any changes both in simulation and on a robot. This approach allows students to be trained with the proper engineering practices and showcases first-hand the advantages and limitations of current robotics simulation tools. We discuss our architecture, which combines the industrial-standard robotics simulator V-REP together with iRobot Create 2 robots, and present the usage of our framework as part of an undergraduate-level course on robotics taught at the University of Southern California—CSCI-445 Introduction to Robotics.

[1]Wolfgang Hönig and Nora Ayanian are with the Department of Computer Science, University of Southern California, USA {whoenig, ayanian}@usc.edu

[2]Arash Tavakoli is with the School of Design Engineering, Imperial College London, UK a.tavakoli@imperial.ac.uk

This work was supported by the Viterbi School of Engineering, University of Southern California, USA

II. RELATED WORK

An overview of robotics for education is given in several books [4], [5]. Unfortunately, simulation tools or the importance thereof are frequently dismissed in the books. Corell *et al.* present a curriculum for teaching robotics using e-Puck robots and the Webots simulator [6]. The simulator allows remote-controlling the e-Puck robot and executing the same code in simulation or on the robot.

Myro [7] is a specialized programming environment for robots in particular for education. It supports several languages and uses robots to motivate non-computer science majors to program. A curriculum [8] for a *flipped classroom* [9] aims to bridge the theory-practice gap in controls education via the use of a *massive open online course* (MOOC) for delivery of lectures [10] and hardware/software platform for practical tinkering. This curriculum offered an in-house robotic simulator—Sim.I.am [11]—based on MATLAB which enables mapping of control code both onto simulated as well as two robotic platforms; namely, Khepera III [12] and QuickBot [13]. In contrast to the aforementioned works, our approach is more generic and can be applied to various types of robots, and does not require any licensing fees for the robotics simulation software used—i.e., V-REP non-limited educational version.

Robotic competitions such as RoboCup [14] highly motivate the use of robotics in education. USARSim [15] was developed to support such competitions and to easily allow to transfer code between simulation and robots. However, the simulator is not widely used outside the RoboCup community.

Surveys of simulation tools specifically for robotics but not tied to education can be found in existing literature [16], [17]. Many different tools exist for various use-cases. GAZEBO [1], V-REP [18], and MORSE [19] appear dominantly in academic conferences for various works on ground robots, aerial robots, and manipulators. Both GAZEBO and V-REP are also frequently used in commercial settings, while MORSE specifically targets academia.

III. REQUIREMENTS

Simulation tools for undergraduate-level education have special requirements compared to those used by professionals only. In particular, the following requirements are important:

Easy Setup and Compatibility. Students must be able to run their code on their own laptops, so that they are able to prepare homework assignments at home and can use the infrastructure beyond the course. Setting up the system should only take a matter of hours, such that it can be done within the first week. Furthermore, the solution should be cross-platform, supporting at least Windows, Mac, and Linux operating systems. Web-based solutions (e.g., [20]) were excluded, since the initial setup is an important part of the learning experience. Virtual machine-based solutions were also excluded for the same reason, and because of poor performance of these solutions in combination with graphics-heavy applications such as 3D robotics simulators.

Simple. The solution needs to be simple in three different dimensions. First, it should be simple to use, so that the students can concentrate on the high-level algorithms rather than low-level details. Second, it should be simple to debug, in particular in simulation, to help students identify and fix bugs quickly. Third, it should be possible for the students to understand the framework itself. This not only makes it possible to extend it beyond the course, but also deepens students' understanding of how the simulation and physical robots differ.

Powerful. Many tools have been proposed specifically for education (e.g., [21], [22]). An undergraduate course should provide sufficiently powerful tools that would be useful beyond the course and can be used as entry-level point into further academic research or industry experience.

IV. ARCHITECTURE

We choose as our simulator V-REP [18], a robotics simulator frequently used in academia and industry. It is cross-platform, free for academic use, and supports various different programming approaches and programming languages to interact and extend the simulator. Installers are available for Windows, Mac, and Linux operating systems and come with all required dependencies, simplifying the initial setup, as required.

Robots in V-REP can be programmed using languages including Java, Python, Lua, MATLAB, and C++. Furthermore, bindings to ROS are available. A survey of our students showed that they have a very diverse spectrum of background knowledge and disciplines. To this end, we decided to exclude ROS and C++ because of their complexity and the difficulty of supporting different compilers on various operating systems[1]. Additionally, we decided to avoid a programming language that requires compilation; students often forget to recompile and therefore the additional compilation step complicates debugging. Thus, to accommodate the aforementioned considerations and to help decrease the programming learning curve for those with little programming experience, we decided to use Python. Its simplicity in design and wide range of available *integrated development environments* (IDEs) allow efficient development and debugging.

For the hardware platform, we selected the iRobot Create 2, a refurbished vacuum cleaning robot specifically designed for *science, technology, engineering and mathematics* (STEM) education [23].

In the following section, we present the hardware, our custom modifications, and our software infrastructure in more detail. The combination of hardware and software allows us to seamlessly integrate simulation and the physical robot.

A. Hardware

The iRobot Create 2 robot is a differential-drive robot with a cylindrical shape with diameter $0.35\,\mathrm{m}$, height $0.1\,\mathrm{m}$, and weight $3.5\,\mathrm{kg}$. Its robust mechanical build can withstand crashes and accidental drops, which is important in classroom settings. The robot can reach a translational velocity of up to $0.5\,\mathrm{m/s}$. The robot's firmware is closed-source; however, a serial port is available, together with an API [24] describing how to control the different actuators and how to read sensors. The following actuators are supported:

- 2 wheels (pulse-width-modulation or velocity control)
- Motors (main brush, vacuum, and side brush)
- LEDs (four binary status indicators and one ring with variable color and intensity)
- Display (four-digit seven-segment)
- Speaker (playing pre-programmed tones)

The following sensors are available:

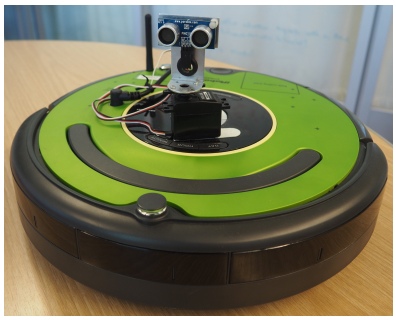
- Wheel encoders
- 6 IR proximity sensors
- 4 cliff sensors (essentially downward facing IR-sensors)
- 3 IR receivers (left, right, omni-directional)
- 2 wheel-drop sensors
- 2 bump sensors
- 8 buttons
- Temperature sensor
- Power-related sensors (voltage, current, and battery capacity)

Furthermore, there is a docking station which sends out IR beams such that the Create 2 can find the dock autonomously for self-charging.

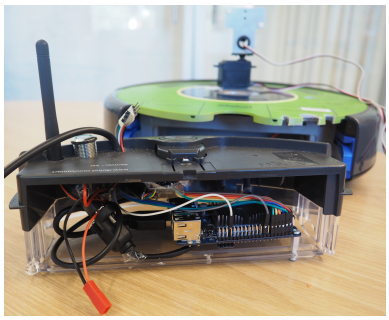
To be able to run software without the need for a cable connection to a laptop, we use the small embedded computer ODROID C1+ [25], running Ubuntu 14.04 LTS. It features an ARM Cortex A5 $1.5\,\mathrm{GHz}$ quad-core CPU, has $1\,\mathrm{GB}$ RAM, boots from microSD, and has four USB ports. We attach a WiFi USB dongle to the ODROID so that students can connect and control the robot wirelessly.

Additional sensors and actuators can be added using the *general purpose input/output* (GPIO) pins. This allowed us

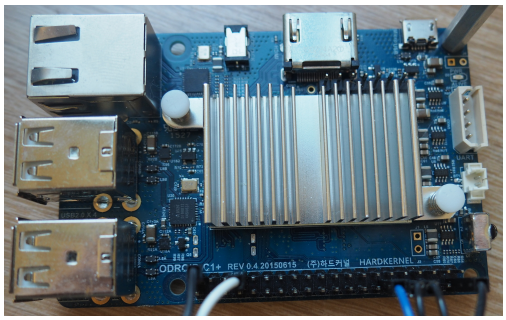
[1]However, we use ROS on the same hardware platform for our research purposes.



(a) iRobot Create 2 with extensions.



(b) Removable extension containing added hardware components.



(c) ODROID C1+ embedded computer.

Fig. 1. iRobot Create 2 robot (a), which we equipped with a small embedded computer (c). Everything is mechanically integrated (b) as a module for easy maintenance.

to extend the robot with an ultrasonic sensor mounted on a servo motor to scan the robot's surrounding.

The ODROID C1+ is directly connected to the main battery, because the extension connector on the Create 2 has a low current limit. We integrate all components into the vacuum container of the robot, making it easier to transport and less affected by any crashes. Furthermore, connectors allow us to quickly swap our hardware additions between robots in order to simplify and speed up maintenance. Our final design, the extension container, and the embedded computer are shown in Fig. 1. The total cost per robot is approximately 350 USD. The list of parts and a wiring diagram are available online as part of the project's documentation[2].

B. Software

We integrated the iRobot Create 2 robot into V-REP as it does not include support for this platform. Specifically, we imported a 3D-model of the robot, added differential-drive joints and wheels, a caster wheel, and the appropriate dynamics parameters to the model. The default V-REP method is to attach scripts to robots, which are written in Lua and include the program logic (so-called child-scripts). Instead, we use the remote API, in which V-REP acts as a server and allows socket connections from any clients (local or remote). The remote API includes only a subset of the functions of the full API, but was sufficient for our use-case. We use the official Python bindings of the remote API.

By default, the simulator controls the simulation loop. In our case, we need full control to mimic the behavior as accurately as possible using the same source code for simulation and physical execution. For that reason we use the *synchronous mode*, in which the next time-step of the simulator can be triggered externally using an API call.

In order to be able to execute the same source code either in simulation or on the robot, we use the *strategy* pattern [26]. We define an abstract interface `Create2` for the robot functionality itself and an interface `TimeHelper` to deal with time related issues. Both interfaces have implementations for the real robot and the simulation. We use the *factory method* pattern [26] to automatically instantiate the desired implementation at runtime. The `Create2` interface provides functions to move the robot or to read sensors. On the physical robot, it uses the serial port of the ODROID and implements the API to achieve the desired behavior. In simulation, the same behavior is created by using remote API functions—e.g., to set the angular velocity of the wheels.

The `TimeHelper` interface provides two functions: `time` and `sleep`, which return the current timestamp and wait for the given amount of time, respectively. In simulation, however, the notion of time is given by the simulator and not by a clock. Depending on the computer and complexity of the simulation, the simulator time might be faster or slower than real-time. We leverage this behavior by advancing the simulator time-steps in our framework, invisible to the user, during the execution of `sleep`. For example, a user can write the following script to move the robot 10 cm forward:

```
create.drive_direct(100, 100) # speed in mm/s
time.sleep(10)
```

On the real robot, the Python script will send a drive direct command to the robot, which will execute for 10 s, until the script exits and stops the robot at this point. In simulation, the first line of the Python script will set the wheels velocities and the second line will advance the simulation steps until the simulator time reached 10 s. Depending on the computer, this might take more or less time compared to the physical experiment.

Additionally, we use a custom run-script to deal with failure cases automatically and in a similar way for both physical robot and simulation. For example, if the script crashes or a user cancels the script, we will automatically stop the robot or end the simulation. The UML diagram of our approach is shown in Fig. 2. The software is available as open-source under the permissive MIT license[3].

In practice, students can first implement an algorithm on their own computers and test it using V-REP by executing:

```
python3 run.py --sim <userScript>
```

Once the results in simulation are satisfactory, the user can copy the scripts to the robot wirelessly and execute

[2] `http://pycreate2.readthedocs.io`

[3] `https://github.com/USC-ACTLab/pyCreate2`

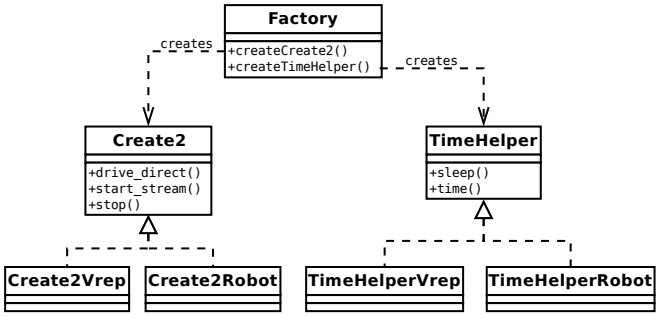


Fig. 2. UML diagram showing the relationship between the key classes of our framework. The `TimeHelper` class is used to deal with the difference between simulation time and real-time transparently for the user.

```
python3 run.py <userScript>
```

on a remote terminal using a *secure shell* (SSH).

C. Discussion

As per our requirements, our architecture is easy to set up, simple to learn and use, and yet powerful. It is easy to set up because it only requires V-REP and Python, both of which come with ready-to-use installers. We found that students can set up the required software and run examples on their own laptops within a 3 h lab session, independent of the operating systems they were using. Using Python as the programming language makes code simple to understand, write, and debug. Students can use IDEs and `print` statements for debugging. Also, we use the CPython interpreter so that a change in the code does not require recompiling. Our framework is written in Python itself, is well documented, and uses standard design patterns, allowing the students to understand and ultimately extend the code for their use-cases. Furthermore, the usage of V-REP, a simulator widely used in academia and industry, allows the students to use their acquired domain knowledge beyond the classroom.

V. CASE STUDY

We used our framework as part of the undergraduate-level course CSCI-445, Introduction to Robotics, at the University of Southern California in Spring 2016. The course requires students to have basic programming knowledge and is open to non-computer science majors. In Spring 2016, we had 30 enrolled students with majors including computer science, electrical engineering, and computational neuroscience. The course consists of lectures and 14 weekly 3 h lab sessions. It covers the foundations of sensors, actuators, control, and motion planning. Lab sessions supplement the lectures by allowing the students to try the concepts covered in practice. In particular, the focus is on understanding the material from the algorithmic point of view, rather than electrical or mechanical perspectives.

For most weeks, students in the lab are split into random groups with three students per team, encouraging teamwork. Each team must demonstrate a working solution in simulation before they are allowed to use the real robots. The lab material is posted at least a day in advance, allowing students to finish the simulation portion of the lab at home, if desired. Nevertheless, all labs are designed such that an efficient team can finish everything within the designated lab time. Students have to document their results on a worksheet. Often they need to reflect their views on the differences they observe between simulation and physical experiments as part of their worksheet, helping them further understand the importance of simulation in robotics, its advantages, and its limitations. Students are also given the full source code of our framework, enabling them to further understand it in depth, and potentially to extend it for their applications outside the scope of the course.

The schedule for the lab is shown in Table I. Most weeks are standalone, but two lab topics span two weeks to allow more complex developments. For example, the midterm project combines the knowledge students learned over the first half of the semester, requiring them to write control software so that a robot can follow specified goal points while avoiding unknown obstacles.

In the following, we outline some of the lab sessions in more detail and explain the importance of simulation during those lab topics.

Sonar Characterization. The students get a V-REP scene with our sonar-mounted robot, but the characteristics of the sonar sensor—e.g., field of view and range—are intentionally specified incorrectly in the simulator. They can use the simulator to verify that their code to read the sonar values behaves correctly. Students then must execute various experiments on the real robot to characterize the sonar, including sensing range, sensing accuracy, field of view, critical angle, and behavior when pointing towards different surfaces. Finally, they are asked to update the simulation model with the obtained parameters and comment on the simulation quality. In V-REP, the sonar is simulated by checking for an object within a visibility cone. In practice, the values are very noisy and depend on the reflective material, angle of incidence, and environment structure—e.g., small gaps in the wall. It is important for the students to understand this limitation so they realize the need for sophisticated algorithms that deal with sensor noise or abrupt measurement variations.

Wheel Encoders. Students must implement odometry to estimate the relative position and heading of the robot using the wheel encoders. First, students use the simulator to verify their implementations; this greatly simplifies debugging since the ground truth values are known and many software iterations can be performed quickly. They must then compare their results in simulation and with real robots in different settings, such as several robot speeds, various wheel encoder update rates, and different types of flooring. The simulation shows similar behavior to the physical robot for different robot speeds and encoder update rates. However, the overall accuracy of odometry is much higher in simulation because some effects such as skidding of wheels or imperfect geometry are not considered in simulation. Having

Week(s)	Topic	Description and role of simulation
1	Introduction	Initial setup of hard- and software. Students learn that the same code can be run in simulation and on the robot with simple move commands.
2	Sonar Characterization	Comparison of sensor behavior between robot and simulation for a sonar. Students learn what typical simulation limits are.
3	Odometry	Let the robot track its position and heading using the wheel encoders. Initial work in simulation simplifies debugging.
4	Odometry Characterization	Comparing the odometry behavior in different cases (e.g., encoder's update rates) for simulation and physical robot. Students learn that corner cases such as skidding are not handled in simulation.
5	Wall Following	Implement a PD controller to let the robot follow a wall. The simulation helps with the initial debugging and tuning, but the real robot has to deal with additional sensor noise and needs to be re-tuned.
6	Go To Goal	Implement PID controller to go to specified goal. The available ground truth in simulation simplifies debugging and accuracy analysis.
7, 8	Midterm Project	Follow waypoints while avoiding obstacles. The simulation environment allows to easily distribute work and improves teamwork.
9	Manipulation	Forward and inverse kinematics of a robotic arm. Students use the simulation to visualize and test algorithms for which no physical platform is available.
10, 11	Particle Filter	Self-localization of the robot in a known environment. The simulator has a dual-use as visualization tool for both simulation and physical setup.
12	Emergent Behaviors	Robotic swarm behaviors. A simplified agent simulator is used, demonstrating how abstraction can help to simulate large quantities of robots.
13	Planning (1)	Search and sampling based planning is introduced. Instead of a simulator, students use different visualization tools to verify the results.
14	Planning (2)	Search based planning is used to dynamically re-plan in unknown dynamic environments. The simulator helps to quickly try and debug various scenarios.

TABLE I. Curriculum of the lab sessions of the undergraduate-level course CS445 Introduction to Robotics thought in Spring 2016 at the University of Southern California. Most weeks require students to demonstrate results in simulation first, before running the code on the robot. Frequently, students are asked to compare the differences between simulation and reality. Some labs span multiple weeks for more complex assignments, where the simulation helps students to parallelize their work.

these results side-by-side demonstrates first-hand such simulation limitations to the students.

PID Controller. Students must implement PID controllers to follow a wall and to go to a specified target position. The simulation significantly aids students in implementing the algorithm correctly. Furthermore, V-REP's integrated graphs can be used to plot data in real-time while the simulation is running, which is less abstract as compared to Simulink-based simulations. Therefore, the trade-off between the P, PD, PI, and PID controllers can be easily explored by investigating the graphs as well as the robot's behavior in simulation. This allows students to gain a practical sense for tuning the controller gains, which is both less time-consuming and less risky in simulation. Finally, the students learn that in order to achieve great performance in practice, the controller gains must be fine-tuned on the physical robot.

Particle Filter. A particle filter can be used to localize a robot in a known environment. Debugging such an algorithm is complicated, as its state (the particles) is large. For example, in a small map used for the class, several hundred particles were required to achieve good self-localization. Here, we provide students with a way of using V-REP not only as a simulator, but also as a visualization tool. This has the advantage that the current state of the visualization and the robots are perfectly synchronized and shown on top of each other. The various particles can be shown as arrows in the same scene. These visualization objects' dynamics are disabled; they therefore do not interact with the physical robot simulation. Once the algorithm works in simulation, the students can execute it on the robot and use V-REP purely for visualization. In order to achieve the desired behavior, we switch to the real-time mode of V-REP and use the remote API directly from the ODROID on-board computer to send the current state to V-REP. This concept is similar to `rviz` in ROS, but does not require additional software to be learned or installed. A screenshot of a simulation session with integrated visualization is shown in Fig. 3.

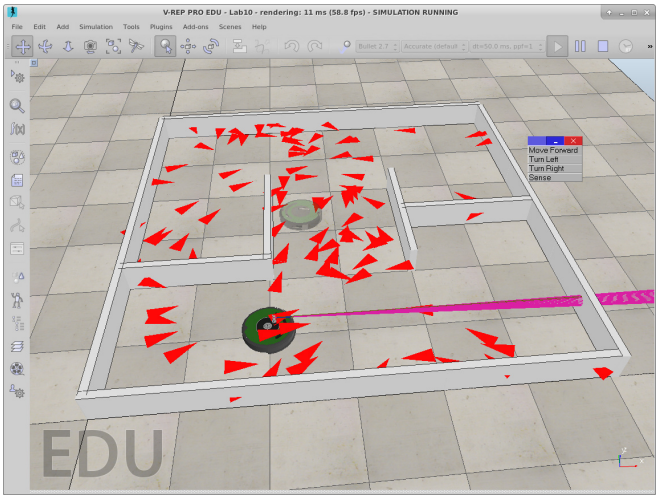


Fig. 3. A test scene for the particle filter task in V-REP. The virtual robot is controlled by a user-written Python script. The various particles are visualized in V-REP as red arrows. The goal is to implement the particle filter correctly such that the robot localizes itself in a known map after a series of movement and sensing actions. The average of the current particles is visualized using a transparent iRobot Create 2 robot.

Planning in Dynamic Environments. Students must implement logic for dynamic re-planning in dynamic en-

vironments. In particular, an initial map is given but additional walls might appear at runtime, blocking one of the feasible paths. The simulation platform greatly simplifies testing, as the walls can be virtually moved while the simulation is running. This allows the user to try many different scenarios quickly, before moving to the physical world.

Manipulator. One lab session includes experiments on a manipulator, which we did not have physically available. However, we used the same framework and Python scripting language to enable students to implement forward and inverse kinematics for serial robotic manipulator arms. Hence, students learned how to develop and test algorithms without access to the physical robots. Implementing the abstraction layer for an actual manipulator would allow the execution of the same code used for the simulation on a physical robot.

Emergent Behaviors. In this lab session, students replicate emergent group behaviors—such as flocking—on a swarm of robots. Researchers in the area of multi-robot teams often do not rely on simulation tools with a physics engine for the verification of their algorithms due to the associated slow runtime for simulations of large groups of robots—e.g., groups of hundreds and above. Instead, agent simulators with a simplified robot model are used. To this end, we also provide students with such a simple simulator and let them discuss as part of their lab assignment what the advantages and disadvantages of the different simulation tools are.

In comparison, the previous version of the course did not use any simulation tools; instead, students executed the code directly on the robot. If the experiment did not work as anticipated, time-consuming trial-and-error experiments on the robot were required to identify whether the fault was software- or hardware-based. In case of any hardware-related issues, the team's progress towards understanding the algorithmic side of the experiments would be stopped until the issues were rectified. Our new seamless simulator integration combines the best of both worlds: debugging fundamental algorithms is simplified, and the same code—without any changes—can be executed on the actual robots, providing experience with physical robotic platforms. This allows interesting side-by-side comparison of the performance of simulated versus physical robots and teaches students how to use and to what extent to trust their simulation results.

Several highly enthusiastic students ended up working on the simulator in their free time, trying tasks which were outside the scope of the course. Similarly, for the labs which spanned over two weeks, students were able to finish the bulk of the work outside the lab, improving teamwork and time management. Finally, we were able to cover more material in the same amount of time compared to the previous year, indicating that the simulation helped students concentrate on the algorithmic aspects of robotics, as desired.

VI. CONCLUSIONS

We present a framework that seamlessly integrates experiments in simulation and on physical robots. In particular, code written in Python can be executed without any changes on either a simulator or a physical iRobot Create 2 robot. Our approach is easy to set up and simple to use, yet powerful since it uses tools used in industry. We demonstrated that such an integrated simulation approach has several advantages in a classroom setting. It simplifies debugging, improves teamwork, allows concentration on high-level algorithmic aspects of robotics, and ultimately allows instructors to teach more material in the same amount of time. Nevertheless, we do not sacrifice real hardware experience, but rather let students experience first-hand what the advantages and limitations of state-of-the-art robotics simulators are.

We believe that our generic approach would be useful in other domains as well. On the education side, the approach can be used in K-12 education and for graduate-level courses. In research, the technique can drastically reduce the time needed for transitioning between simulation and implementation, for reduced training time of new lab members, or as tool for robotics researchers with backgrounds outside of computer science.

REFERENCES

[1] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004, pp. 2149–2154.

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

[3] A. Behrens, L. Atorf, R. Schwann, B. Neumann, R. Schnitzler, J. Balle, T. Herold, A. Telle, T. G. Noll, K. Hameyer, and T. Aach, "Matlab meets lego mindstorms—a freshman introduction course into practical engineering," *IEEE Transactions on Education*, vol. 53, no. 2, pp. 306–317, 2010.

[4] D. P. Miller and I. R. Nourbakhsh, "Robotics for education," in *Springer Handbook of Robotics, 2nd Ed.* Springer, 2016, pp. 2115–2134.

[5] B. S. Barker, G. Nugent, and N. Grandgenett, *Robots in K-12 Education: A New Technology for Learning*. IGI Global, 2012.

[6] N. Correll, R. Wing, and D. Coleman, "A one-year introductory robotics curriculum for computer science upperclassmen," *IEEE Transations on Education*, vol. 56, no. 1, pp. 54–60, 2013.

[7] M. M. McGill, "Learning to program with personal robots: Influences on student motivation," *ACM Transactions Computing Education*, vol. 12, no. 1, pp. 4:1–4:32, 2012.

[8] J. P. de la Croix and M. Egerstedt, "Flipping the controls classroom around a MOOC," in *American Control Conference*, 2014, pp. 2557–2562.

[9] B. Tucker, "The flipped classroom," *Education next*, vol. 12, no. 1, 2012.

[10] Coursera, "Control of mobile robots," https://www.coursera.org/learn/mobile-robot, website accessed: Nov. 2016.

[11] J. P. de la Croix, "Sim.i.am," http://gritslab.gatech.edu/projects/robot-simulator, website accessed: Nov. 2016.

[12] K-Team, "Khepera iii," http://www.k-team.com/mobile-robotics-products/old-products/khepera-iii, website accessed: Nov. 2016.

[13] O'Botics, "Quickbot," http://o-botics.org/robots/quickbot/, website accessed: Nov. 2016.

[14] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *International Conference on Autonomous Agents*. ACM, 1997, pp. 340–347.

[15] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 1400–1405.

[16] M. Torres-Torriti, T. Arredondo, and P. Castillo-Pizarro, "Survey and comparative study of free simulation software for mobile robots," *Robotica*, vol. 34, no. 4, pp. 791–822, 2016.

[17] A. C. Harris, "Integration of the simulation environment for autonomous robots with robotics middleware," Ph.D. dissertation, The University of North Carolina at Charlotte, 2014.

[18] E. Rohmer, S. P. N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326.

[19] G. Echeverria, S. Lemaignan, A. Degroote, S. Lacroix, M. Karg, P. Koch, C. Lesire, and S. Stinckwich, "Simulating complex robotic scenarios with MORSE," in *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, 2012, pp. 197–208.

[20] The Construct Sim LTD, "The Construct," http://www.theconstructsim.com, website accessed: Nov. 2016.

[21] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[22] University of California, Davis, "CSTEM Studio," http://c-stem.ucdavis.edu/studio/, website accessed: Nov. 2016.

[23] iRobot Corporation, "iRobot Create2," http://www.irobot.com/About-iRobot/STEM/Create-2.aspx, website accessed: Nov. 2016.

[24] ——, "irobot create2 open interface (oi) specification based on the irobot roomba600," http://www.irobotweb.com/~/media/MainSite/PDFs/About/STEM/Create/iRobot_Roomba_600_Open_Interface_Spec.pdf?la=en, website accessed: Nov. 2016.

[25] Hardkernel co., Ltd, "ODROID Documentation," http://odroid.com/dokuwiki/doku.php?id=en:odroid-c1, website accessed: Nov. 2016.

[26] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995.