# A Remote/Virtual Robotics Lab

R. Pito Salas (rpsalas@brandeis.edu)
Julian Ho (julianho@brandeis.edu)
Brandeis University, Waltham, Massachusetts, USA

***Abstract*—In this Work-In-Progress paper, we describe our vision for a remote/virtual robotics lab and our progress towards realizing it. Robotics has been taught at our institution for four years with enrollment growing each year. Our program is part of the undergraduate Bachelor of Science in Computer Science and has an equal focus on robotics theory and practice, with the "Robot Operating System" (ROS) running on students' own computers, and a small robotics learning lab with a variety of ROS based robots. Our pandemic experience during 2020 led us to invest in a cloud-based shared "cloud" environment running on a Kubernetes cluster. We learned that there are real advantages to this in terms of scale and geographical reach. The success of this with our students, our Covid experience, preparation for teaching in our lab, and planning towards return to campus led directly to this work-in-progress paper, where we define, in one abstraction, a remote/virtual robotics learning lab, unifying local and remote access to both simulated and real robots. This work aims to extend our existing shared virtual cloud-based robotics learning environment and combine it with a remote-controlled robotics lab.**

## I. Introduction

At the start of 2020, as the Covid pandemic forced us to shutter our Robotics Lab, like many, we rushed to create a virtual environment to continue teaching remotely. This has been referred to as "Robots as a service"[8]. We were successful in creating a Kubernetes[2] cluster with the requisite software installed, and accessible by a web browser. This was a monumental task and taught us a lot. That experience combined with planning for our return to campus led us to ask why our virtual robotics environment could not be married to our physical robotics lab, having both available either locally or remotely.

### A. Motivation and previous work

Hands-on experiential learning greatly enhances and is arguably required for learning the fundamentals of robotics. This argument is strongly supported, for example, in "Best Practices in Robotics Education"[6], which makes a strong case for the importance of hands-on work in robotics education. To provide this experience, we have two choices: either each student has a robot of their own, or we provide students access to the robotics learning lab. Both of these approaches work, but have drawbacks. Robots issued to students are expensive, delicate, and can often be challenging for students to work with. A robotics lab is an excellent solution, but only scales in proportion to its size.

A remote-controlled robotics lab could allow much greater scale as well as remote access. Pickem et al. [4] describe the Robotarium, a remotely accessible, multi-robot research facility which was one of the first to experiment with these ideas. Tzafestas et al. [7] describe a virtual and Remote Laboratory that studies these issues. Think of this as a laboratory with an API. Using this API, robots can be allocated, initialized, prepared, and placed in real physical spaces. And the student can watch this streaming live. Naturally, the true hands-on experience is lost, but we gain considerable scale and cost benefits, in addition to accessibility to remote students.

What would provide the most effective learning environment? A robotic lab for students to sign up and use, or a set of simulated robots in simulated environments? Are we satisfied with requiring students to be physically present or do we want more geographical flexibility? Would it use ROS? Michieletto et. al. [3] argue (and we agree) that ROS may not be the best pedagogical entry point for undergraduate students.

Fig. 1. Real/Virtual and Local/Remote

| | Real Lab and Robot | Simulated Robot and World |
|---|---|---|
| Local Access | ✓ | ✓ |
| Remote Access | 🤔 | ✓ |

We looked at all permutations of real/virtual and local/remote and asked ourselves whether a single model could describe all four quadrants (see Fig 1). While, as described above, each of the quadrants exists, to our knowledge, no one has attempted to unify them into a single model.

To be specific, a remote/virtual robotics lab is accessed via a web interface which provides a standard programming environment consisting of a command prompt and a text editor. A session can be started virtually, where the student selects a type of robot, a virtual world, and runs their program and sees the results. Or, with the same interface, schedule a real-world session, again selecting a type of robot, a physical world, run their program, watch the actual robot, and see the results over streaming video.

We argue that if it is possible to unify the virtual and real, local and remote, the cognitive and learning load on students is reduced. Algorithms can be designed and tested in simulation and then easily tested in a lab without students having to have to be on campus, or having to learn a completely different environment.

These are the questions we aim to answer with this work:

**Q1** Can the virtual, physical, local, and remote robotics labs actually be unified into a single consistent abstraction?

**Q2** If so, will this lead to better student learning?

### *B. Requirements*

Based on our research and experience, we arrived at the following key requirements for a remote/virtual robotics laboratory:

1) Accessibility: The lab should be accessible and usable by students from any kind of computer, running any operating system. Requiring a high-end computer would limit accessibility.
2) Remote Use: Provisioning and use should be possible without requiring physical presence. Students should be able to set up and be productive without ever setting foot on campus.
3) Programming Environment: The Lab should provide an easy programming environment where students can build on their knowledge and use well established, open and not proprietary tools.
4) Scale: The Lab should allow classes of as many as 100 students to use its resources simultaneously and asynchronously. In addition, scale requires that adding new students is an automated process, not requiring expert access to each student's computer.
5) Simulation: The Lab should provide access to basic Robot simulations, including models of robots and different premade worlds.
6) Coordinated use of Lab robots: The Lab should provide for remote "time-shared" access to physical robots in the Lab.
7) Cost: Additional cost to the student, should not exceed the cost of an expensive textbook.

## II. Structure and status

Our research model is illustrated at a very high level (see figure 2 below):

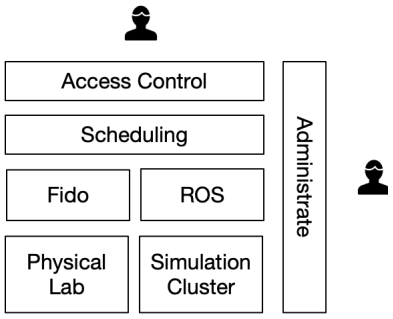


Fig. 2. High level architecture

- Access control: provides login and authentication, as well as authorization, resource quotas and capabilities.
- Scheduling: Manages scheduling sessions, assigning and changing time slots, allocating and freeing resources.
- ROS: The standard ROS software stack used both in the physical and virtual environments.
- Fido: A simplified robot-agnostic library designed for teaching and non-robotics researchers.
- Physical Lab: An actual lab with one or more physical robots and one or more physical worlds ("arenas", such as mazes). A lab may or may not have a staff member monitoring it.
- Simulation Cluster: A cluster with as many computers as needed to run the simulations and other lab management software.
- The administration layer allows control and management of the whole system.

### *A. Sample Use Case*

To make the vision concrete, here is a possible use scenario: Students log on through the access control and schedule a robot and a space. For the virtual lab, scheduling allows spreading out the use of the limited resources of memory, cores, and persistent storage. For the physical lab, scheduling ensures that the arena and robot being reserved are available.

Resource permitting, students gain access through their web browser to a programming environment and visualization of the robot and the space. Once the session begins, the student will see the world (either simulated or streamed) as they write and test their algorithms, which are coded either using ROS or using a simpler software layer for non-ROS programmers we call Fido.

Administrators (teachers and teaching assistants) log on through the administration interface where they can manage accounts, virtual robots, and worlds that are available as well as real robots and labs that are available. Staff can be scheduled as well for the lab as needed.

## III. Simulation environment - the Virtual Lab

Our virtual lab is a simulation environment with the following additional characteristics. It provides accessibility using a simple browser, without requiring a particular web browser, operating system or other specific software preinstalled on the students' computer. And within the virtual lab environment, it provides a programming environment for students to write and test their programs. This environment further provides a variety of simulated worlds, with one or more robots and a choice of models.

### *A. Server Cluster*

We have developed software running on a set of servers, orchestrated via Kubernetes, with all requisite ROS and other software preinstalled. This cluster has been running in production for the last six months with great success.

### *B. Provisioning and Access Control*

Provisioning is the process whereby new students can be given access easily and without affecting other students. An individual student may have different resource requirements, so it is also necessary to be able to allocate additional core or GPU resources to individual students. During normal work, sometimes the environment has to be updated with new course scaffolding or totally reset after a corruption.

## IV. Remote controlled physical Lab

This is the most speculative aspect of the remote/virtual robotics lab and the one that we have not yet prototyped. The lab envisioned could be located anywhere on or off campus, and does not have to be staffed constantly. What is unique is that it is under the control of our software.

### A. Automation

Through the scheduling interface, a student can reserve a time and desired "arena" (for example, a maze, a track marked by cones, etc.). Available time slots and configurations are defined administratively. During the allocated time, a top-down camera would stream video to the student. At the end of the session, the robot would be fully reset to a known state, ready for the next student.

### B. Lab Operations

The above concept is quite open ended. A specific initial implementation, which is quite feasible, is to allocate a block of time (possibly at night while the lab is not being used) with a fixed single available configuration (e.g., a 5x5 meter arena, surrounded by blocks) and a single physical robot (e.g. a Turtlebot3). These rules mean that we don't have to build a new lab, we can just use our existing lab during off hours with existing robots.

Initially, this lab would have to be staffed to manually collect robots that are stuck, as well as recharge batteries that have run down. However, over time both of these needs could be automated.

## V. Programming Model

We also wish to address the ways in which students and researchers actually write and experiment with their algorithms.

As stated before, ROS is a standard environment for serious roboticists. We provide a robust and complete ROS programming model (editor, shell and all installed packages) to control either a simulated robot in the virtual lab mode or a real robot in the remote lab mode.

In our experience, however, there is a need and desire for non-robotics researchers to use robots as part of their work. They are not software engineers nor roboticists. To address this important use-case, we have merged Fido (unpublished so far) as a first-class programming model alongside ROS. Fido is a pure Python, non-ROS programming environment and model aimed at this much larger set of potential users so as to test the proposition that our vision may extend beyond serious roboticists into other disciplines. The work on Fido is well under way.

### A. ROS

Unlike some similar projects, our ROS support is mainstream and unmodified. Students get a complete and conventional command shell, with a particular version of ROS installed and an available text editor.

### B. Fido Programming Environment

As mentioned above, we are also designing a simpler programming environment called Fido for students and researchers who do not need ROS. Fido is inspired by Pyro [1] and Jyro [5], two general robotic programming abstractions which were designed specifically for teaching undergraduates.

Our goal is that Fido scripts will be able work on both ROS and non-ROS robots, both simulated and real. Fido is delivered in the form of a well-documented Python library with a simple programming model that is designed with the non-roboticist in mind. It is necessarily less powerful than ROS, but it is also far more accessible.

Here (see below, figure 3) is an example of a simple Fido script which shows how robot control and simulation control are both captured in one script. Unfortunately, there is no

```
from fido.robot import Turtlebot3
from fido.simulation import Gazebo, Simul
from fido.world import RaceTrack

robot = Turtlebot3("bot_01")
world = RaceTrack()
world.add(robot, x=0, y=0, z=0)

sim = Simulation(
    simulator=Gazebo(gui=True),
    world=world,
)

sim.start()

# Move at speed 2.0 for 5.0s
robot.move(speed=2.0, duration=5.0)
robot.stop()

# Rotate for 45 degrees.
robot.rotate(angle=45.0, speed=2.0)

sim.stop()
sim.destroy()
```

Fig. 3. Sample Fido Code

room in this brief report to delve into the details of Fido. It is currently working but still early in its development.

We do want to underline two categories of challenges that Fido faces: first, unifying ROS and non-ROS robots under one abstraction. In other words, the possibility to write single script that could be used on robots running different operating systems. This challenge has not, as far as we can determine, been addressed,

The second challenge is unifying the control of the robots with control of the simulation environment – that is to say, a single programming model, where one script could control any combination of real vs. simulated, and ROS vs. non-ROS robot. This challenge has been addressed from time to time, for example with Jyro [5]. Jyro is an especially well thought out

hardware-independent simulation environment. However, for the remote/virtual robotics lab, we are specifically requiring a real-world physical robot counterpart to our simulated robot.

## VI. Status

As stated, our remote/virtual robotics lab is in the earliest stages. It was inspired by our work on a virtual robotics teaching lab to meet very specific needs in our response to Covid. All our work is public and others are welcome to experiment and contribute to it. Here is the current state of the project:

1) The simulation cluster is proven and is well documented and in production use. It consists of docker and Kubernetes scripts and images, automation for provisioning and rebuilding, and more. It is set up in a way that other newer versions of ROS could easily be provided.
2) The Fido programming environment is in progress. Simulation environments are brought up and simple Fido scripts can control robots within them. However we still need to prove the control model for more complicated scenarios and support non-ROS robots.
3) Access Control is currently implemented via web based log in which takes the student to a shell and a program editor with which algorithms can be coded and run in simulation. Administration commands exist for managing users, resetting students' environments, and other similar tasks

## VII. Conclusion

In this work-in-progress paper, we described a remote/virtual robotics lab consisting of a virtual and real environment, accessible both locally and remotely. Our insight is that a single abstraction could be designed to embrace these modalities and lead to a simpler and more flexible teaching platform. Much remains to be done for us to measure and demonstrate the effectiveness of this approach. Our immediate priorities are a) to bring Fido to a usable level for students and b) to create a prototype of the remote-controlled lab so that we can further prove the validity of this approach.

## References


[1] Douglas Blank et al. *Advanced Robotics Projects for Undergraduate Students*. Tech. rep. URL: www.aaai.org.

[2] Kubernetes.org. *Kubernetes, Automated container deployment, scaling, and management*.

[3] Stefano Michieleto et al. "Why Teach Robotics using ROS?" In: *Journal of Automation, Mobile Robotics and Intelligent Systems* 8.1 (2014), pp. 1–23. ISSN: 1923-0265. DOI: 10.14313/JAMRIS.

[4] Daniel Pickem et al. "The Robotarium: A remotely accessible swarm robotics research testbed". In: *Proceedings - IEEE International Conference on Robotics and Automation* (Sept. 2017), pp. 1699–1706. ISSN: 10504729. DOI: 10.1109/ICRA.2017.7989200. arXiv: 1609.04730. URL: http://robotarium.org/%20http://arxiv.org/abs/1609.04730.

[5] Becky Tang and Alex Robey. "Autonomous Robotic Navigation Using Self-Organized Maps". In: (2017), pp. 1–10. URL: https://www.cs.swarthmore.edu/$%5Csim$meeden/cs81/f17/projects/AlexBecky.pdf.

[6] Shawna Thomas. "Best Practices in Robotics Education: Perspectives from an IEEE RAS Women in Engineering Panel [Women in Engineering". In: *IEEE Robotics and Automation Magazine* 28.1 (2021), pp. 12–15. ISSN: 1558223X. DOI: 10.1109/MRA.2021.3051833.

[7] Costas S. Tzafestas, Nektaria Palaiologou, and Manthos Alifragis. "Virtual and remote robotic laboratory: Comparative experimental evaluation". In: *IEEE Transactions on Education* 49.3 (Aug. 2006), pp. 360–369. ISSN: 00189359. DOI: 10.1109/TE.2006.879255.

[8] Chongkun Xia et al. "Microservice-based cloud robotics system for intelligent space". In: *Robotics and Autonomous Systems* 110 (2018), pp. 139–150. ISSN: 09218890. DOI: 10.1016/j.robot.2018.10.001. URL: https://doi.org/10.1016/j.robot.2018.10.001.