Open Remote Web Lab for Learning Robotics and ROS With Physical and Simulated Robots in an Authentic Developer Environment

Dāvis Krūmiņš, Sandra Schumann, Veiko Vunder, Rauno Põlluäär, Kristjan Laht, Renno Raudmäe, Alvo Aabloo, and Karl Kruusamäe

***Abstract*—Teaching robotics with the robot operating system (ROS) is valuable for instating good programming practices but requires significant setup steps from the learner. Providing a ready-made ROS learning environment over the web can make robotics more accessible; however, most of the previous remote labs have abstracted the authentic ROS developer environment either for didactical or technological reasons, or do not give the possibility to program physical robots. In this article, we present a remote web lab that employs virtual network computing and Docker to serve in-browser desktop workstations, where learning tasks can be completed on both the physical and simulated robots. The learners can reserve access to the remote lab through a learning management interface, which also includes tools for administering the remote lab. The system allows anyone to experiment with ROS without configuring any software locally and was successfully trialed in an online ROS course.**

***Index Terms*—Docker, educational robotics (ER), remote robot operation, robot operating system (ROS) education, virtual network computing (VNC), virtual robotics lab.**

I. Introduction

In RECENT times, the robot operating system (ROS) has become the *de facto* framework for software development in robotics [1]. This is mostly in part due to its choice not to impose many limitations on the developers, who are able to browse a large collection of existing software libraries, build modular code in a programming language of their choice, and integrate devices from different manufacturers. Learning robotics with ROS ensures the employability of obtained skills and seamless transfer from the context of learning to the workplace [2].

One reason why learning a new technology—such as robotics with ROS—can be considered challenging is that the target technology is simultaneously in two roles: 1) the content and 2) the mediator [3] of the learning experience. In robotics, the learner is often expected to use a specific robot with its software framework to learn both the particular tools and the wider concepts of robotics. Thus, the learner needs a functional robot and its development environment to learn robotics. Here, the biggest value of the ROS as a universal and modular software framework poses a challenge of the initial system setup either on the learner or on the educator. The effort of the system setup often involves installing Linux—a potentially new operating system for the learners—along with the necessary ROS and robot-specific software packages [4]. Even when clear step-by-step instructions are available, the setup can be intimidating and discouraging to uninitiated learners. Furthermore, the setup may require administrator-level access to the learner's computer, and any oversight during the process can possibly lead to a corrupted system and loss of personal data. The access to necessary equipment (i.e., robots, sensors, and compute resources) is often another hurdle on the learning journey. Educational institutions usually possess only a limited amount of physical robots due to their relatively high cost. A self-guided lifelong learner may find themself limited to low-cost alternatives with a wider technological skill gap from learning to professional setting.

To increase accessibility and accommodate a larger number of students, an alternative is to teach ROS and robotics with the help of simulators such as Gazebo [5]. Learning via simulation can be beneficial, because the student does not have to travel to any specific location to access a robot, and even for on-site learning, we can scale the learning output beyond the limitation set by the number of available physical robots. Despite the widespread adoption of simulation in robotics education in recent years, the instructors generally report reduced learning quality when there is no access to physical equipment [6]. Moreover, the use of simulation does not necessarily eliminate the burden of setup as the learner is often still expected to install new software on their personal computer.

Remote web labs [7] are one technology to potentially accommodate both access to physical robots and simulation environments while offering minimal setup and maximizing adaptive personalized learning experience. In fact, remote labs are being identified among the most valued education technologies, especially in the field of computer engineering [8].

Despite several remote lab solutions being piloted in the domain of robotics [9], [10], [11], [12], [13], [14], [15], [16],

Manuscript received 6 May 2023; revised 3 December 2023 and 2 March 2024; accepted 20 March 2024. Date of publication 27 March 2024; date of current version 9 April 2024. This work was supported in part by the Education and Youth Board of Estonia, in part by European Social Fund via IT Academy program, in part by the Estonian Centre of Excellence in IT (EXCITE) funded by the European Regional Development Fund, and in part by the AI and Robotics Estonia co-funded by the EU and Ministry of Economic Affairs and Communications in Estonia. *(Corresponding author: Dāvis Krūmiņš.)*

The authors are with the Institute of Technology, University of Tartu, 50090 Tartu, Estonia (e-mail: davis.krumins@ut.ee).

This article has supplementary downloadable material available at https://doi.org/10.1109/TLT.2024.3381858, provided by the authors.

Digital Object Identifier 10.1109/TLT.2024.3381858

[17], [18], the vast majority aim to simplify (either for didactical or technological reasons) the environment in which the learning occurs. To ensure minimal context change from the learning to professional setting, which is often the primary motivation to those interested in acquiring skills in ROS robotics, a remote web lab that reflects the authentic professional development experience is needed.

In order to facilitate engaging student learning and eliminate the tedious environment setup that newcomers face when entering the world of ROS, we propose an open-source web system where students can access readily available development environments, which offer the possibility of interactively programming both the simulated and physical ROS robots. Such a system allows students to skip the acquisition of ROS software or robots and start their robotics learning journey with a realistic robotics development environment.

The contribution of this work can be summarized as follows.

1) We developed a technical solution for hosting remote web labs for learning robotics. The system leverages virtual network computing (VNC)-based graphical desktop-sharing and Docker containers together with a dedicated web app that authenticates learners and allows them to book time on virtual Linux machines. In this remote web lab, learners can gain access to physical and simulated robots via a web browser over an Internet connection.
2) The system was validated in an online course about ROS and robotics (191 participants), where it enabled learners to complete practical tasks of operating and programming robots in an authentic Linux and ROS environment.
3) The source code and setup instructions for the presented remote web lab are publicly available[1] for allowing other educators to host it on their servers and integrate with their robots. We have verified the system with a Robotont mobile robot [19] and a UFACTORY xArm7 manipulator robot (Video S1 of the Supplementary Material).

II. Requirements

In order to develop the requirements for an authentic remote web lab that enables learners to be immersed in the context of ROS developers, we sketch the key components of the professional workplace. A full-stack ROS developer typically works on a computer with the Linux operating system, which is connected to a physical robot. In addition to the ROS and robot-specific software, the developer is empowered by a wide set of (software development) tools, e.g., IDEs and version control systems. Very often, the software setup also includes some form of robot simulation, so that there is possibility to debug and test the program code without always powering on the physical robot.

While an aspiring ROS developer can replicate this setup on their computer, it does require a certain level of technical skill, particularly in the domain of operating systems and their administration. Obtaining and exhibiting such skills may at times be part of the intended learning outcomes, but it can also serve as a barrier to the true objective, which is to learn the concepts of ROS and robotics.

From the literature, we can conclude that to accommodate the widest possible target audience while ensuring learning outcomes applicable in real life, an optimal learning environment for robotics and ROS would need to meet the requirements listed in Table I [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35].

III. Related Work

In an attempt to overcome barriers of robotics education, several remote web labs have been reported [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], but the vast majority compromise with an abstracted web UI, do not offer physical robots, or require the user to set up complementary software on their own computer. Here, we discuss and compare (see Table II) three works that stand out in terms of meeting the requirements outlined in Table I: Robot Learning Lab (RLL) [15], Construct Sim [16], and OpenUAV [17].

A. Robot Learning Lab

One of the most sophisticated remote labs providing access to physical robots through the web is RLL [14]. The general offer of RLL is an open-source fully automated code submission and processing system for multiple Kuka LBR iiwa 7 R800 robot arms. The users of RLL can enter the job processing queue by trying out one of the available demo programs or by submitting their own code written in general-purpose programming languages, such as C/C++ and Python. The robot executing the user's submission can be viewed live in the browser, and once the program is finished, log files and a video recording are stored and made available to the user for downloading. RLL is a great platform for researchers; however, it might not be ideal for robotics newcomers since the web interface only provides results of a task, and the primary workflow must be done on the user's own computer. The setup involves configuring an API access key in a ready-to-use virtual machine or manually installed RLL software [36], which does not solve the problem of quickly jumpstarting the learning journey of a novice.

B. Construct Sim

The Construct Sim is a commercial robotics-as-a-service web-based platform for learning ROS online and provides both simulation environments and physical robots to its subscribers [16]. The Construct fulfills many of the prior requirements as the custom ROS development environment is fully integrated in the web browser. However, the custom-widget-based UI is an imitation of a desktop workspace and contains abstractions to the natural ROS developer environment; consequently, users do not get the chance to gain the valuable experience of working with one. The Construct offers both robotic manipulators and mobile robots with a vast collection of courses that also covers ROS2 topics, but as closed source software, it cannot be used in cases where teaching staff would want to design a unique course

[1]Replication package. [Online]. Available: https://doi.org/10.5281/zenodo.7769596 GitHub. [Online]. Available: https://github.com/unitartu-remrob/remrob-server

TABLE I
REQUIREMENTS FOR A REMOTE WEB LAB FOR LEARNING ROS ROBOTICS

	Requirement	Justification
1	Minimal setup time for uninitiated learner: • no need to have administrator level access to the learner's computer; • cross-platform in terms of operating systems.	Setting up a new operating system, ROS, and physical robots can be very intimidating and has a high risk of data loss when done by a non-expert. The time required to make the software functional takes away from cognitive engagement, and instructors are forced to spend resources on technical support [20]. The student may not know if all the effort is worth it in the beginning of the learning journey [21]. Lowering the entry barrier by minimizing the effort of setup allows professional robotics tools (e.g. Linux and ROS) to become an educational tool for robotics [22], [23].
2	Possibility to validate learning tasks on physical robots.	Evidence shows that there is no significant difference in the learning outcome between groups using simulated or physical robots [24]. However, in an attitude survey [24], the group using physical robots displayed a larger interest in the topic; participants reported that they felt more engaged and confident in writing programs than the robot simulator group. Similar findings have also been reported by other studies [25]-[27]. Furthermore, it is valuable for students to understand the limitations of simulated environments [28].
3	Possibility to streamline development with robot simulation.	The opportunity of using simulation often streamlines software development when the learner is still in the phase of structuring the code and developing the basic components of the algorithm. The use of physical robots is more time consuming and introduces the risk of encountering hardware problems [27], [29]. Learning the concept of simulation, a widely adapted tool in robotics shows how to develop programs quickly at a lower cost in a risk-free environment [30].
4	The learning environment would reflect the authentic real-life environment to ensure the transfer of skills from learning space to professional setting.	An argument can be made that an abstracted learning interface can initially be less intimidating for a beginner [18]. Nonetheless, when learning takes place in a virtual environment that simplifies the actual workflow of developing robotics software, it may create false or unrealistic expectations of the related professional work as the obtained skills are context-bound [31], [32]. For efficient transfer of knowledge and skill the learning experience should "hug" the target performance [33], i.e. be as close a representation of it as possible. It has also been observed that authentic online learning improves students' perceptions about their employability competences acquisition [34].
5	Open and customizable.	Deploying open-source solutions in education can have many benefits such as higher customization in terms of available features, smaller risk of service being discontinued, continuous improvement by a like-minded community, and absence of license fees [35].

TABLE II
FEATURE COMPARISON OF EXISTING REMOTE ROBOTICS WEB LABS

	Requirement	RLL	The Construct	OpenUAV
1	Minimal setup time for the uninitiated learner	-	+	+
2	Physical robots available	+	+	-
3	Development with robot simulation	+	+	+
4	The learning environment fully reflects the real-life environment	+	+/-	+
5	Open and customizable	+	-	+

environment with their own custom hardware and materials in a language tailored to a specific audience.

C. OpenUAV

OpenUAV is an open-source platform that provides out-of-the-box Lubuntu ROS development environments, which are served through a web-based VNC interface and, thus, mimic a natural ROS workstation [17]. The VNC approach is flexible since the programs in use do not need WebGL libraries for browser adaptation, e.g., *Gzweb* for Gazebo [37] being used in the first OpenUAV iteration [38]. The simulation-based system employs cloud infrastructure to achieve the required computational power for photorealistic robotics simulations and is only missing the addition of physical robots to fulfill the requirements we have put forward.

IV. FUNCTIONAL OVERVIEW

The remote lab was designed for use in an online ROS course, which anyone should be able to complete without having to worry about whether their computer is suitable for working with robots. Accordingly, the system has the following attributes.

1) *Web access:* Browser-based access to a learning environment that represents the authentic Linux OS a ROS developer uses. The use of a web application ensures minimal setup effort for the learner, no need for administrator-level access to the learner's computer, and cross-platform usability.
2) *Physical robots:* The learning environment is preconfigured to connect with a physical mobile robot.
3) *Support for simulation:* The learning environment can also be used to validate ROS programs in a simulation environment (e.g., Gazebo).
4) *Multiple test beds:* The learning environments are available simultaneously for multiple learners.
5) *Live stream:* The user's robot can be observed via the same web interface from which it is being controlled.
6) *Learning management:* Issuing of either simulation or physical robot environments is a streamlined process that can also be managed through an administration interface.
7) *Session persistence:* The work done by users does not disappear and is automatically available for their next session.

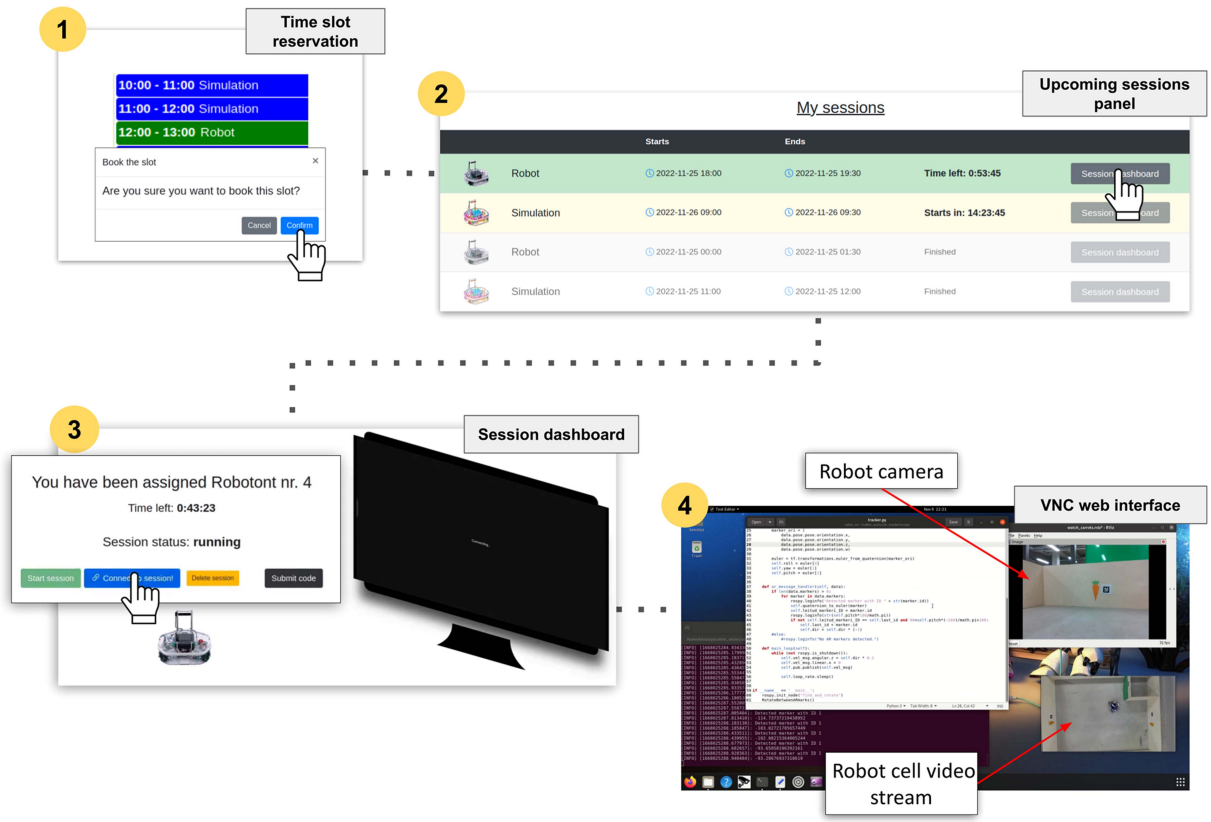

Fig. 1. Routine steps a user follows to interact with a physical robot: 1) a time slot is reserved in the calendar; 2) session becomes active at a specific time; 3) in the session dashboard, the user can manage their personal ROS environment; and 4) the environment is available in the user's browser window.

A. User Perspective

In the Supplementary Material, we provide a video demonstrating use of the remote lab (Video S1 of the Supplementary Material). The showcased web application has the following two distinct parts:

1) *staging area:* a custom-built session reservation and management interface with the purpose of authorizing users for access to their workstation;
2) *workstation:* a web VNC view of a Linux machine prepackaged with ROS.

The staging area consists of a minimalist interface that attempts to offer its users an intuitive way for establishing a web VNC session to run robotics simulations or interact with a physical robot remotely. Access control is implemented via predetermined time slots, which administrators are responsible for creating (Video S2 of the Supplementary Material). The time slots are listed in a calendar, each belonging to one of the "Simulation" or "Robot" categories (see ① in Fig. 1). The users can decide which of the available times is most suitable to them and claim a slot (Video S3 of the Supplementary Material). With the activation of a session, users gain access to a dashboard (see ③ in Fig. 1), from which it is possible to boot their workstation and connect to it. Once the session time runs out, the user is automatically disconnected to make space for the next learner. In the staging area, the learner has the chance to:

1) reserve a session tied to a specific time slot in the calendar;
2) see their past and current session information in the booking table;
3) boot their workstation once a session is initialized;
4) reset the workstation to the last state;
5) commit the workstation's current state to memory, i.e., save the changes in its file system;
6) view whether the robot assigned to them is online.

The workstation is a VNC view of the Ubuntu 20.04 operating system running the default GNOME desktop. The provided environment contains a camera application to view the live stream of the assigned robot cell from a top–down perspective. To make sure that the learners can continue with their previous work, the included ROS workspace containing all the external and user-written packages is persistent across the user's sessions.

To guarantee quality of service, an administrator should be present in the remote lab. To successfully run the remote lab, they are empowered with tools that aid in learning management. The administrator of the remote lab can use the staging area to:

1) activate or disable user accounts;
2) define the time slots during which users can access robots or simulation environments;
3) declare which cell contains which robot, swap, or disable any individual one.

Any technical problems encountered by the user when they are present in the remote lab need to be troubleshooted by the administrator. The process is facilitated by an administrative

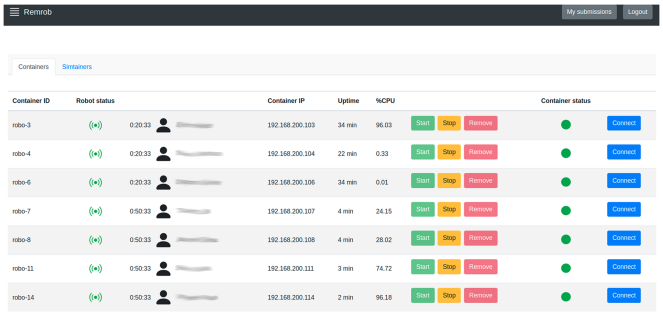

Fig. 2. Administrative panel for monitoring user activity.

panel, which allows monitoring of the active user sessions (see Fig. 2).

With the help of the panel, the administrator can:

1) determine if a user has activated their session;
2) find out which robot a user has been equipped with;
3) monitor the status of the robot, whether it is present in the network;
4) see every session's expiration time;
5) monitor how much computational resources are being consumed by each user;
6) start or stop a specific session;
7) connect to a live user's session.

V. System Architecture

Our full system architecture is depicted in Fig. 3. The components of the implemented interactive learning system can be roughly divided as follows.

1) *Virtualization components:* These components are the base of the user's workstation—a virtualized learning environment with all the necessary software for working with ROS.
2) *Access provision:* The access provision components are parts that make up the staging area and help automate creation of user workstations and assignment of robots. They contain tools for user authentication, authorization, and administration of the remote lab.
3) *Linking elements:* They are components that are crucial for gluing together the different parts of the system.

A. Virtualization Components

The role of virtualization components is in creating a well-defined working environment that resembles a real desktop computer and can also be easily wiped clean once a user disconnects.

1) Docker: Open-source container technology Docker, a lightweight operating-system-level virtualization, is used to provide users with isolated sandbox environments for testing and solving problems without fear of corrupting the primary operating system. With Docker, resource allocation is done directly by the host operating system's kernel, which makes containers more resource effective compared to virtual machines [39]. Containers can be customized for each user with session-specific environment variables and launched quickly. Another benefit of using Docker for the creation of learning workstations is that it enables simultaneous user connections on a single multitenant server.

2) TigerVNC: To interactively access the ROS workstation over the web, a VNC server implementation known as TigerVNC is incorporated into the Docker container [40]. It runs "headless" by using a virtual framebuffer and has optimizations for 3-D and video applications, while also providing inbuilt configuration programs including *vncpasswd* that assists in enabling temporary session authentication. To provide a display that TigerVNC can use, the GNOME desktop environment was also integrated into the Docker container, making its appearance closer to that of a regular Linux virtual machine.

3) VirtualGL: Open-source simulation software Gazebo and the sensor data visualization tool RViz are both widely used by ROS developers. These components depend upon OpenGL API, which is responsible for rendering 2-D and 3-D graphics on a graphical processing unit (GPU). However, VNC clients cannot access the remote server's hardware and by default are not capable of server-side hardware-accelerated rendering [41]. VirtualGL solves this problem by redirecting the OpenGL 3-D rendering commands (GLX calls) to the host server's hardware (see Fig. 4). The TigerVNC server compresses the rendered 3-D images into 2-D frames, which can then be sent over the network to the VNC client as usual. Hardware acceleration provides a much smoother user experience and offloads the rendering burden from the central processing unit (CPU), which leaves more resources for regular robotics data computations.

GPU-accelerated containers are made possible by CUDA and OpenGL Docker images released by NVIDIA [42], which can be utilized through the NVIDIA Container Toolkit [43]

B. Access Provision

The main purpose of access provision is to establish a secure path for the user to the VNC server, so they can interactively work with their robot. The access provision components constitute the staging area, which helps in the automation of learning management tasks.

1) noVNC and Websockify: To access the prepackaged ROS learning and development containers via web, the open-source VNC JavaScript client noVNC is employed. So far, the project has been actively maintained and runs in any modern web browser [44]. It uses the WebSocket protocol, which is the standard for full-duplex real-time web applications. However, TigerVNC does not have support for WebSockets as of yet, and thus, a WebSocket to TCP socket proxy *Websockify* is utilized to translate traffic coming and going to the TigerVNC servers [45]. *Websockify* is capable of proxying multiple connections at once, and the target VNC server can be distinguished by the request URL parameter.

2) Container Control API: While automated issuing of already running containers is feasible, for administrative purposes, it is beneficial to have remote container management available. Giving regular users the privilege to restart or save their environment whenever they desire is also an appealing prospect. To accomplish it, an API with capabilities to monitor, start, restart, and commit a container state was written for the *NodeJS* runtime,

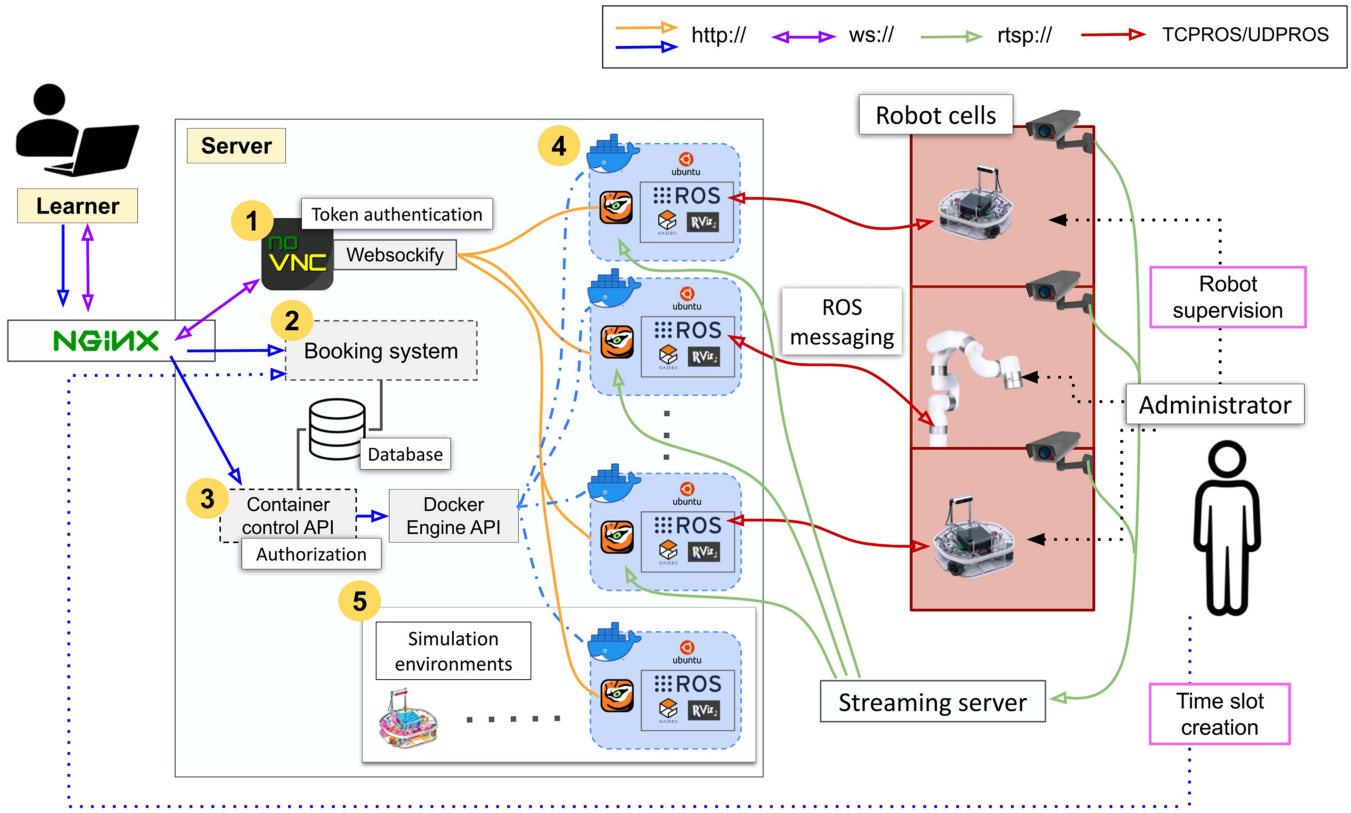

Fig. 3. System overview. The main software components of the remote robotics lab server are as follows: 1) noVNC web client; 2) session reservation; 3) container management; 4) robot-enabled containers; and 5) simulation containers.

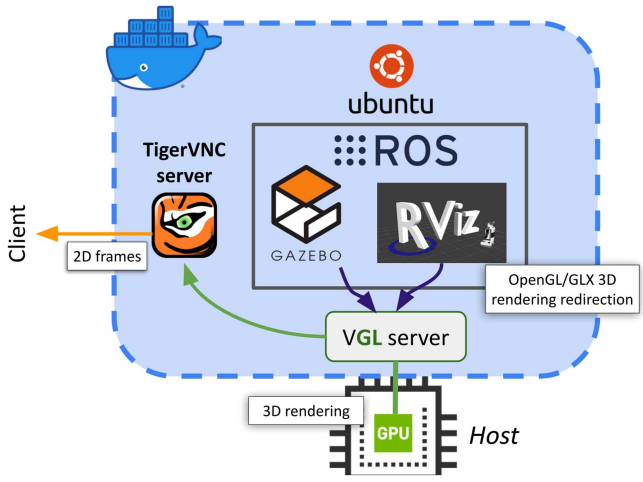

Fig. 4. Hardware-accelerated 3-D graphics rendering within the Docker container.

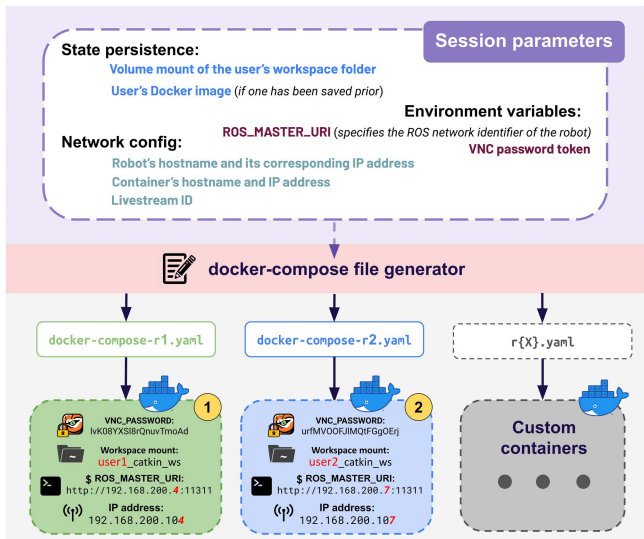

Fig. 5. Launching custom containers with the help of autogenerated *docker-compose* files. The containers can be customized to the specific user while taking into account the robots available.

where the node package manager registry provides a wrapper library called *Dockerode* that helps simplify communication with the Docker Engine API [46].

We introduce a method for preparing the container with a user-specific session configuration. Fig. 5 visualizes how the desired container state can be coherently organized with the help of *docker-compose*. When a request is sent for a container to be initialized, a *docker-compose* YAML configuration is automatically generated including:

1) the specification of the user's personal ROS catkin workspace folder to be mounted for saving the work of a session;

2) an environment variable containing a randomly generated token that is consumed by the Docker entrypoint script to set the password for the VNC server. The same token is included in the request response for establishing a connection via noVNC;
3) (physical robot session) the assigned robot's IP address to be included in the container's bash profile, where it is used to identify the robot as the ROS master host;
4) (physical robot session) a static IP address for the container itself which the assigned robot is configured to communicate with;
5) (physical robot session) the identifier of the cell the assigned robot is located in for the purpose of providing the correct live stream.

Each of the containers is highly customizable for the individual session, which would be much harder to achieve with a virtual machine.

3) Booking System: The amount of physical robots and computational resources that can be provided is limited, while the number of users who wish to access the system can exceed those limits. To organize live access to the remote lab, a booking system was created, where the time slots that are made available to students are submitted by an administrator (Video S2 of the Supplementary Material). When a session becomes active, the user will get assigned a free container, which they are authorized to operate until the session time runs out.

C. Linking Elements

The networking setup plays a key role in ensuring that the multirobot remote lab is functional. With only a single Docker host, each of the containers must be linked to both the mobile robot and the client on the other end of the network. The virtualization and system access components also must be consolidated to form a deployable web application.

1) Container–Robot Networking: In the ROS, communication is established via XML remote procedure call, while the data themselves are sent over TCPROS or UDPROS protocols [47]. Nodes in the ROS environment each run on their own port, which means that for unobstructed communication, there should be full bidirectional access on all the ports among the engaged network units. Accordingly, the requisite to expose the Docker containers to the robot network was apparent. The Docker *macvlan* network driver is suitable for this purpose. If the physical network interface is able to function in the promiscuous mode, i.e., is capable of handling multiple MAC addresses, then a *macvlan* Docker network driver can distribute packets to their corresponding containers. The process effectively gives each container a unique LAN IP address and provides multiple container–robot pairs that can be given out to users simultaneously.

2) Nginx: Singular entry point into the system is guaranteed by *nginx*, a versatile open-source web server capable of efficiently proxying WebSocket connections [48]. All the access provision components, that is, the noVNC application, container control API, and the booking system, are made available on their own respective URL paths.

VI. Results

The solution described in Sections IV and V has been fully implemented, and the source code is freely available online (see footnote 1). Table III analyzes how the resulting remote web lab meets the requirements listed in Table I [49], [50].

The web application with simulation environments can be acquired for experimentation through an automated Docker installation.

The general steps for a manual replication to host the remote web lab are further outlined here.

A. Server Hardware

In choosing server hardware, consider that with a more powerful processor, more users can be served at once and with better performance. A dedicated NVIDIA GPU is required for getting faster processing speed in 3-D visualization tasks.

B. Software Setup

1) Install Ubuntu 20.04 and the required software on the server.
2) Set up container management.
   a) Acquire the Docker image that users will use as their workstation.
   b) Build and run the container management back end.
   c) Run a script to generate compose files used for booting containers. Hardware acceleration can be enabled by modifying the templates.
3) Build and run the staging back end and front end.
4) Consolidate components by implementing the prepared *nginx* configuration.

C. Addition of Physical Robots

1) *Lab network:* Both the server and the robots are placed in the same network.
2) *Server routing:* Each of the containers is given a unique network identifier for standard ROS communication. The server is configured to mask itself as multiple devices by utilizing the Docker *macvlan* network driver.
3) *Robot network configuration:* For communication that includes hosting a ROS master on the robot, the container's hostname resolution is specified on the robot's computer.
4) *Documenting robots:* The container control back end is supplied with the IP addresses of the robots provisioned. The appropriate configuration files must be changed accordingly (see online materials at GitHub).
5) *Live streaming:* The camera stream of the remote lab cell can be implemented in: (a) the staging area, (b) the user's workstation, or (c) a third-party streaming application.

VII. Validation in an Open Online Course

In order to validate that the proposed remote web lab can truly enable delivering ROS and robotics education on a large

TABLE III
FULFILLMENT OF REQUIREMENTS

	Requirement	Validation
1	Minimal setup time for the uninitiated learner.	The developed remote web lab can be accessed through a web browser. By using a graphical user interface to book a time slot (Video S3), the learner can gain access to a remote desktop environment in which ROS is readily available (Video S4). The learner is not expected to install any additional software on their local computer as long as they have a web browser and internet connectivity.
2	Physical robots available for validating the learning tasks.	Access to the robot is achieved by a direct network link between the mobile robot and the container. When the in-built camera application is opened a top-down view of the robot is seen (Video S4).
3	Possibility to streamline development with robot simulation.	Two types of containers are currently available in the developed system: 1) virtual machines with access to physical robots and 2) virtual machines with only simulation capabilities. The Gazebo simulation software is preinstalled in both types of virtual machines. When access to physical robots is limited or not needed for the particular learning task, the learners and educators can leverage more accessible simulation-only containers.
4	The learning environment would reflect the real-life environment to ensure the transfer of skills from learning space to professional setting.	The environment learners face is almost an equivalent of an Ubuntu 20.04 Focal Fossa GNOME desktop environment. Ubuntu is the most common and recommended operating system for ROS software development [49], [50]. There is no additional application developed on top of the default Ubuntu environment, and learners are guided to use it the same way as would a professional ROS developer.
5	Open and customizable.	The source code is available on https://github.com/unitartu-remrob/remrob-server, it can be freely modified under the MIT License. The software setup steps are documented in the code repository.

TABLE IV
OVERVIEW OF THE LEARNING OUTCOMES FOR THE COURSE MODULES

	Learning Outcomes	Homework Assignment
M1	The learner navigates between directories, edits files and opens applications in Ubuntu 20.04; uses simpler Linux command line commands; moves the robot to a desired position via teleoperation.	Teleoperating the robot via command line, motion visible by overhead camera.
M2	The learner distinguishes ROS nodes, topics, messages; operates and constructs *rosrun* and *roslaunch* commands.	Remapping nodes from several different pre-existing packages to make the robot autonomously drive between orange markers.
M3	The learner designs a digital description of a robot using Unified Robot Description Format (URDF); modifies a URDF description using *xacro*.	Creating a URDF description of a four-wheeled robot, moving it in RViz using keyboard teleoperation.
M4	The learner prepares a new ROS package with an executable node; designs a program to make the robot move in a predetermined static pattern using ROS publishers.	Writing a publisher to make the robot move in a square, circular and figure 8 shaped static trajectory.
M5	The learner extracts info from sensors using ROS; designs a ROS subscriber and a simple controller; constructs a launch file.	Writing a simple controller that makes the robot turn between AR markers.
M6	The learner performs simultaneous localization and mapping; autonomously navigates a robot to a desired location using ROS Navigation library.	Mapping the environment, autonomously navigating a robot to a desired location.

scale, we deployed and tested it during a massive open online course (MOOC), held in Estonian from September to November in 2022 [51]. The objective of the MOOC was to provide an introduction to programming robots using ROS and achieve the learning outcomes listed in Table IV. The participants accessed robots only over the remote web lab to complete six modules (see Table IV) over the course of seven weeks. In total, 191 participants signed up for the MOOC that did not assume any prerequisite knowledge in Linux, ROS, or programming.

The 2 ECTS course was self-paced during seven weeks, and the recommended pace for course participants was to complete one module per week (with an extra week as a buffer). Each module consisted of four parts:

1) theoretical background;
2) practical ROS-based exercises;
3) an automated self-test;
4) an independent homework exercise.

Theoretical background and instructions for ROS-based exercises were published on the public course website.[2] The practical exercises in ROS required that the participants use the remote

[2][Online]. Available: https://sisu.ut.ee/rosak

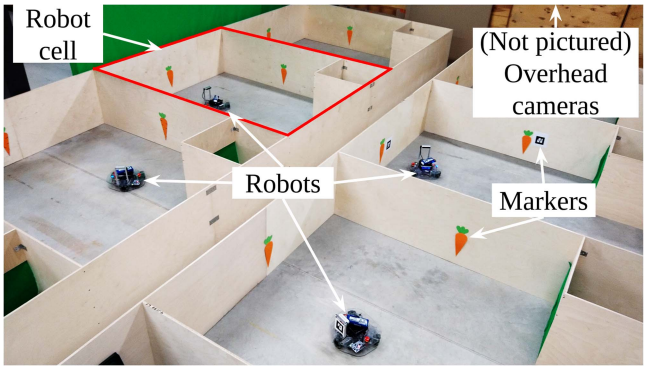

Fig. 6. Classroom used during the ROS MOOC.

web lab as their main tool. At the end of each module, the participants completed a theory-based test and a practical homework exercise (see Table IV). The homework exercise was completed using the remote web lab.

A. Web Lab Infrastructure Setup

In order to deliver the course and enable the remote web lab, a dedicated physical infrastructure was created to facilitate the system depicted in Fig. 3. A plywood grid of eight robot cells each containing a single mobile Robotont robot [19] was constructed (see Fig. 6). Each cell was equipped with an overhead camera that was streaming video to the participants. In addition, the robot cells had colorful and augmented reality (AR) markers on their walls for use in various tasks.

Each robot was connected to a robot-enabled container. In addition, nine simulation containers were made available to the participants. Each participant created a remote web lab account and could then book a session to either use a robot-enabled container or a simulation container, depending on their needs and task at hand. The available session lengths varied from 25 to 90 min.

Here, we lay out the technical setup details specific to the launched MOOC.

1) *Hardware:* The system was deployed on a server with Intel i5-12600KF CPU, 32-GB RAM, and NVIDIA GeForce RTX 3060 Ti GPU.
2) *Lab Network:* MikroTik RB962 hAP 802.11a/b/g/n/ac serves as the gateway router, while an AXE5400 Tri-Band WiFi 6E (802.11ax) router connected to the RB962 is used for establishing fast 5-GHz band wireless connections to the mobile robots, which are equipped with the Intel Dual Band Wireless-AC 8265 Wi-Fi adapter.
3) *Live streaming:* Two 12th-generation Intel NUC streaming servers each hosting four Logitech C922 web cameras for a total of eight streams providing view of every separate robot cell (see Fig. 7). Each of the camera streams is proxied by a WebRTC streamer application running on the main server [52]. During a session, the corresponding stream of the assigned robot was made available in the VNC view of the user's workstation. The user could launch the stream through a desktop shortcut, which opens the specific WebRTC endpoint in a Chrome web browser window, where it can be toggled to a floating Picture-in-Picture video (see ④ in Fig. 1).

B. Methodology

To assess the value of the remote web lab platform, the following three forms of data were collected:

1) metrics describing the learning progress;
2) voluntary feedback from the participants of the course;
3) the time logs about use of the remote web lab system.

The voluntary feedback questionnaire consisted of 25 questions divided into six sections. One of the sections consisted of questions about the usability of the system: the system usability scale (SUS) [53] and an open-ended question about the web lab environment. The collected data were analyzed using descriptive statistics.

C. Results and Discussion

Course participants came from a variety of backgrounds. The youngest person to sign up for the course was 14 years old and the oldest 72. The youngest person to complete all the six modules was 16 years old and the oldest 72. 40% of the survey respondents reported having a master's degree or equivalent qualification, 26% a bachelor's degree or equivalent, and 18% a secondary school diploma as their highest attained level of education. 73% of those who signed up for the course were male and 27% female, and among those who completed the course 82% were male and 18% female. 70% of the survey respondents reported having IT or technology as one of their fields of specialty.

The analysis of learning metrics shows that out of the 191 participants that signed up for the course, 101 completed the first module, i.e., they successfully created an account, booked a time slot, accessed the remote Linux desktop during the allotted time, and interacted with a physical Robotont robot (see Fig. 8). In total, 49 learners went on to complete all the modules of the course, i.e., completing the six different tasks (see Table III) using the remote web lab.

Although a 25% turnout might seem low at first, it is not a worrying metric for an MOOC course in general as it has been observed that course completion is not the goal of many students who sign up for MOOCs, and it is not uncommon to witness turnout rates close to 10% [54], [55]. Inclusion of more "gamified" content in course modules could potentially drive higher student engagement and consequently lower student attrition [55].

Fig. 9 depicts the results from the SUS questionnaire. The mean score was 77.76 ($N = 48$), which corresponds to the evaluation of "good" when compared to the database of other products [56]. Unsurprisingly, the scores were higher among participants who completed more modules of the course: the mean score was 83.36 among participants that completed all the six modules ($N = 29$), compared to 72.50 among those that completed some, but not all the modules ($N = 15$), and 56.88 among those that did not report having completed any modules ($N = 4$). It is unclear; however, whether the user experience with

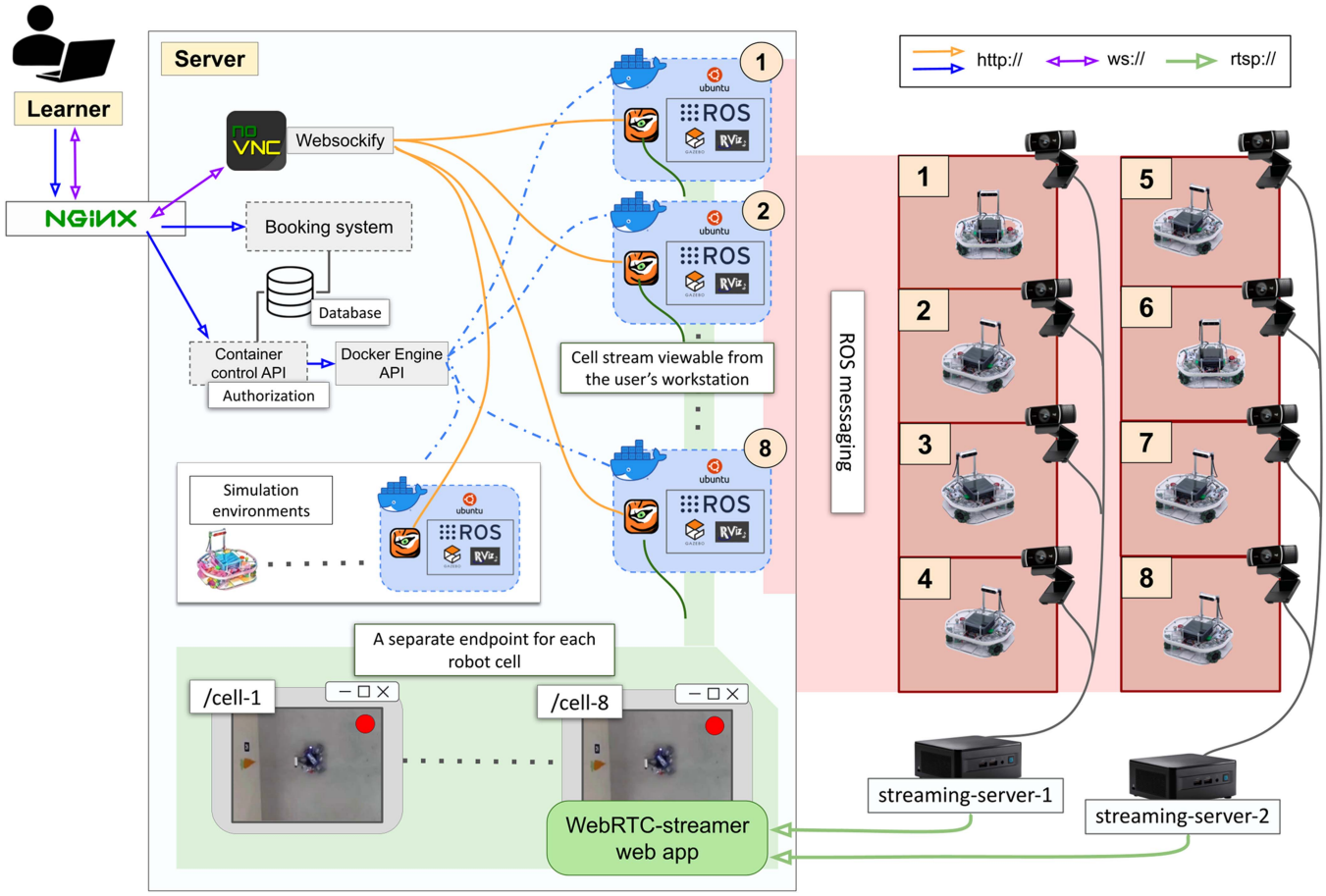

Fig. 7. Robot cell video streaming setup for the ROS MOOC including the use of web cameras.

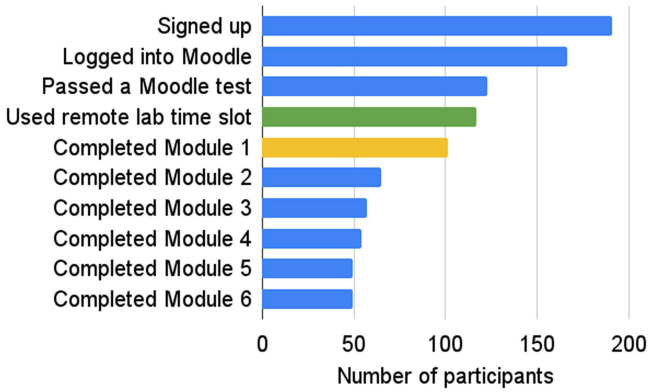

Fig. 8. Participants' progress through the course.

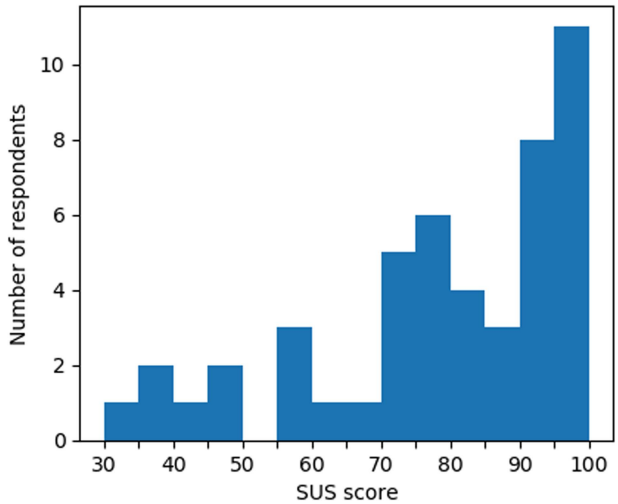

Fig. 9. Histogram of SUS results.

the remote web lab played a significant role in not completing the course.

Based on the remote web lab usage data, those who completed all the course modules spent on average 8.06 h using the robot-enabled containers and 23.8 h using the simulation only containers. Based on the feedback survey, those who completed all the six modules reported having spent 6.36 h per week on the course on average, and those that did not complete all the modules spent 4.12 h per week.

The course infrastructure was observed to be mostly dependable. In order to prevent congestion, the video feed transmitted from robots to the server required compression or throttling, or for the image processing to be done on board. These solutions proved effective. The time delay between the users' containers and robots was negligible in most cases and did not significantly

impact performance. However, when the participants' personal Internet connections were unreliable, performance was notably affected.

The course capacity was limited by hardware availability, as the robots could not autonomously recharge their batteries and required the presence of an instructor. However, efforts are underway to develop this capability for the specific robotic platform used.

VIII. Performance

To investigate the performance and potential capacity (i.e., the optimal amount of containers to schedule for access), we conducted a series of load tests to see how the server load impacts the frame rate of graphical visualization and framebuffer update rate on the client side, which can be linked to the perceived smoothness and responsiveness of the remote desktop experience.

A. Experimental Setup

The hardware and software setups during the performance tests are as follows.

1) *Server:* Intel i5-12600KF CPU, 32-GB RAM, NVIDIA GeForce RTX 3060 Ti GPU, and TigerVNC v1.10.0.
2) *Client:* Apple MacBook Pro M2 16" using Google Chrome browser v122 with noVNC client v1.4.0 and window resolution of 1920 × 1015 pixels; the image quality and compression were set to the default settings (6/9 and 2/9, respectively). Since the goal of the test is to see how the server load affects user experience, the client network speeds were throttled at 20 Mbit/s for download and 10 Mbit/s for upload; latency was set at 30 ms.

Two sets of experiments were conducted: increasingly loading the server with containers running 1) physics-based Gazebo simulation and 2) RViz-based visualization. The first case would imitate learners completing tasks in a simulation-only mode and is more performance intensive as the server needs to provide computation for the physics engine. The latter case with RViz simulation would more closely resemble working with a robot where part of the computation is not handled by the server but by individual onboard computers of the robots.

A video example of a Gazebo performance test is available in the Supplementary Material (Video S5 of the Supplementary Material).

B. Results and Discussion

The Gazebo test (see Fig. 10(a) and (b)] can be considered as the worst case scenario as its physics simulation is more resource intensive than a container with RViz visualization [see Fig. 10(c) and (d)]. Gazebo is also more graphically rich in detail as can be seen when comparing framebuffer update rate between Fig. 10(a) and (c). Given the current hardware and software configuration, we observe that CPU load reaches saturation with five parallel containers running Gazebo simulation [see Fig. 10(b)] or eight parallel containers running RViz [see Fig. 10(d)]. This is further confirmed from the usability perspective as the Gazebo

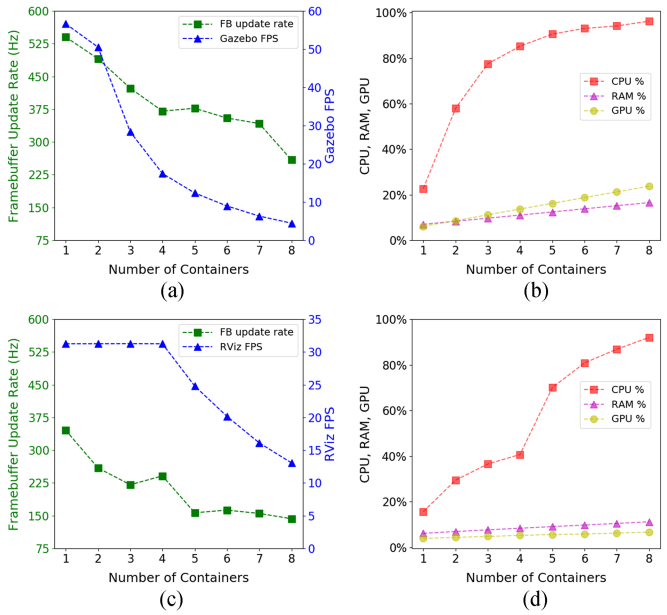

Fig. 10. (a) Client performance for Gazebo simulator. (b) Server load for running Gazebo containers. (c) Client performance for RViz visualization. (d) Server load for running RViz containers.

frame rate drops below 10 frames/s (FPS) when more than five containers are working in parallel.

However, it should be noted that during a scheduled session, the learner is not expected to constantly run the simulation or work on a robot. Time is also spent on writing code, compiling it, and studying learning materials—thus, the server load gets naturally balanced. If, on average, half of the session is spent engaging with a simulation, then eight containers would bring average FPS values closer to that of a server under the load of four containers. It is also true that learners progress at different speeds; learners who finish their tasks faster free up computational resources for others in the shared time slot. In addition, they might be at different lessons altogether, some using the less CPU consuming RViz and others working only with the Linux command line (see Table IV). To prevent a single learning from potentially burdening the whole system, we suggest implementing resource constraints on the container, which can be enabled in the compose template (see Fig. 5).

The noVNC application does not autoadjust the compression and image quality for the client's network bandwidth [44]. While changing the default values for the learner can be done by modifying its source code, an alternative is to instruct the learners to increase the compression and lower the image quality in the noVNC viewer based on the quality of the network they are using.

IX. Discussion

To gain insights on how students perceive the remote lab's environment, conducting further questionnaires or interviews with course participants could be beneficial. This would help determine if the ability to start robotics exploration without initial setup steps was an appealing prospect, whether students

believe that the acquired skills are immediately applicable in real-world scenarios. Devising a study with two distinct groups of students each using either only simulated or physical robots for learning task completion could contribute more data to the literature on the differences between learning effectiveness between simulated- and physical-robot-based education.

A. Use Cases Beyond Education

The system was built with educational goals in mind; however, it can also be used by professionals who wish to run simulations on more powerful hardware than what their own personal computers can offer. The proposed web application is also useful for times when the researcher is not able to be physically present but still desires to run a test with a physical robot. Another potential use case is trialing new robots remotely before making a decision on whether to buy or rent one.

B. Use Cases Beyond Robotics

The remote lab can be thought of as a Linux-server-based computing infrastructure supplemented with robots and cameras. High-performance computing (HPC) is becoming increasingly important for natural sciences [57], yet a survey has reported that a substantial portion of scientists (40%) prefer to run simulations on their own computers regardless of the enormous wait times [58]. A crucial obstacle is that access to HPC systems is largely command-line based, while most students and researchers are more comfortable working with graphical interfaces [59]. The remote lab introduced is a virtual desktop infrastructure solution not limited to robotics simulations; therefore, any other software for physics, chemistry, or biology computational sciences can be incorporated into the workstation's container. Despite Linux not being as crucial for these applications as for ROS, our system could potentially provide a way for scientists to utilize the server's powerful hardware to view 3-D visualizations, e.g., a protein fold, from a personalized workstation that they can access over the web from anywhere.

An example of web-based containerized graphical desktop sharing being implemented for bootstrapping portable customizable workstations is GUIdock-VNC [60]. As a cloud-compatible tool built on top of Docker and noVNC, it simplifies collaboration for bioinformatics workflows and ensures study result reproducibility.

C. Limitations

1) Cybersecurity: Although measures were taken to safeguard the booking system, container control API, and the live workstations against unauthorized access, the freedom given to the user in operating their ROS container can prove dangerous. Docker containers have less isolation from the host due to the kernel they share with it, a characteristic that makes them less secure than virtual machines. Moreover, to enable GNOME desktop within the containers, additional SYS_ADMIN privileges were granted, which expose even more attack surface for a potential container breakout that can result in data leakage or denial of service [61].

During the system trial, all the containers dedicated to pairing with the physical robots were freely operating in the same network as the main server. To prevent foreign connections, i.e., a container pairing with a robot not explicitly assigned to it, firewall rules should be configured on either the server or the router. For more stringent network isolation and protection of the server's network resources, a more sophisticated Docker infrastructure incorporating proxy containers could prove valuable; the authors of RLL describe this safety measure in more detail [14].

2) Performance: During the operation of the remote lab, the maximum number of simultaneous users was capped at nine, because only a single physical server was instituted. Beyond that limit, the computational resources would get stretched thin, and as a consequence, the quality of the user experience could drop significantly. Horizontal server scaling with load balancing software for the operation of multiple physical servers should be considered for increasing the system's capacity.

D. Future Improvements

The system could also be used in conjunction with ROS2, but currently no adjustments have been made to support it. The process would consist of introducing the appropriate ROS2 software to the robots and the base Docker image, together with adding appropriate namespacing for each of the identical robots.

Another useful upgrade would be to introduce automatic charging stations [9]. This would reduce the time lab administrators must spend in swapping the discharged batteries of the mobile robots.

X. Conclusion

In robotics, there are significant barriers of entry to newcomers, whether it be hardware or the required software. The widely adapted robotics software development framework ROS is a valuable starting ground for students but has a markedly steep learning curve. In this article, we propose and implement a remote web lab that eliminates the obstacle of a time-consuming environment setup. Our web lab allows students to quickly attempt their hand at learning the concepts of robotics and ROS in a realistic development environment. Both the physical and simulated robots are made available at defined time slots that students can book in advance. The remote web lab is trialed in an MOOC geared toward introducing robot programming with ROS to a general audience.

References

[1] M. Quigley et al., "ROS: An open-source robot operating system," vol. 3, 2009.

[2] S. Michieletto, S. Ghidoni, E. Pagello, M. Moro, and E. Menegatti, "Why teach robotics using ROS?," *J. Autom., Mobile Robot. Intell. Syst.*, vol. 8, pp. 60–68, Jan. 2014, doi: 10.14313/JAMRIS_1-2014/8.

[3] M. Bower, "Technology-mediated learning theory," *Brit. J. Educ. Technol.*, vol. 50, no. 3, pp. 1035–1048, May 2019, doi: 10.1111/bjet.12771.

[4] S. Pietrzik and B. Chandrasekaran, "Setting up and using ROS-kinetic and Gazebo for educational robotic projects and learning," *J. Phys. Conf. Ser.*, vol. 1207, Apr. 2019, Art. no. 012019, doi: 10.1088/1742-6596/1207/1/012019.

[5] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators," in *Towards Autonomous Robotic Systems*, M. Giuliani, T. Assaf, and M. E. Giannaccini, Eds., Cham, Switzerland: Springer, 2018, pp. 357–368.

[6] A. Birk and D. Simunovic, "Robotics labs and other hands-on teaching during COVID-19: Change is here to stay?," *IEEE Robot. Autom. Mag.*, vol. 28, no. 4, pp. 92–102, Dec. 2021, doi: 10.1109/MRA.2021.3102979.

[7] E. G. Guimarães, E. Cardozo, D. H. Moraes, and P. R. Coelho, "Design and implementation issues for modern remote laboratories," *IEEE Trans. Learn. Technol.*, vol. 4, no. 2, pp. 149–161, Apr.–Jun. 2011, doi: 10.1109/TLT.2010.22.

[8] S. Martin, E. Lopez-Martin, A. Moreno-Pulido, R. Meier, and M. Castro, "The future of educational technologies for engineering education," *IEEE Trans. Learn. Technol.*, vol. 14, no. 5, pp. 613–623, Oct. 2021, doi: 10.1109/TLT.2021.3120771.

[9] B. Pitzer, S. Osentoski, G. Jay, C. Crick, and O. C. Jenkins, "PR2 remote lab: An environment for remote development and experimentation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 3200–3205, doi: 10.1109/ICRA.2012.6224653.

[10] P. Petrovic and R. Balogh, "Deployment of remotely-accessible robotics laboratory," *Int. J. Online Eng.*, vol. 8, pp. 31–35, 2012.

[11] M. Kulich, J. Chudoba, K. Kosnar, T. Krajník, J. Faigl, and L. Preucil, "SyRoTek—Distance teaching of mobile robotics," *IEEE Trans. Educ.*, vol. 56, no. 1, pp. 18–23, Feb. 2013.

[12] G. A. Casañ, E. Cervera, A. A. Moughlbay, J. Alemany, and P. Martinet, "ROS-based online robot programming for remote education and training," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 6101–6106, doi: 10.1109/ICRA.2015.7140055.

[13] D. Pickem et al., "The Robotarium: A remotely accessible swarm robotics research testbed," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 1699–1706, doi: 10.1109/ICRA.2017.7989200.

[14] M. S. D. S. Lopes, I. P. Gomes, R. M. P. Trindade, A. F. da Silva, and A. C. D. C. Lima, "Web environment for programming and control of a mobile robot in a remote laboratory," *IEEE Trans. Learn. Technol.*, vol. 10, no. 4, pp. 526–531, Oct.–Dec. 2017, doi: 10.1109/TLT.2016.2627565.

[15] W. Wiedmeyer, M. Mende, D. Hartmann, R. Bischoff, C. Ledermann, and T. Kroger, "Robotics education and research at scale: A remotely accessible robotics development platform," in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 3679–3685, doi: 10.1109/ICRA.2019.8793976.

[16] R. Tellez, "A thousand robots for each student: Using cloud robot simulations to teach robotics," in *Robotics in Education* (Ser. Advances in Intelligent Systems and Computing), vol. 457. M. Merdan, W. Lepuschitz, G. Koppensteiner, and R. Balogh, Eds., Cham, Switzerland: Springer, 2017, pp. 143–155.

[17] H. Anand et al., "OpenUAV cloud testbed: A collaborative design studio for field robotics," in *Proc. IEEE 17th Int. Conf. Autom. Sci. Eng.*, 2021, pp. 724–731, doi: 10.1109/CASE49439.2021.9551638.

[18] W. A. M. Fernando, C. Jayawardena, and U. U. S. Rajapaksha, "Developing a user-friendly interface from robotic applications development," in *Proc. Int. Res. Conf. Smart Comput. Syst. Eng.*, 2022, pp. 196–204, doi: 10.1109/SCSE56529.2022.9905084.

[19] R. Raudmäe et al., "ROBOTONT—Open-source and ROS-supported omnidirectional mobile robot for education and research," *HardwareX*, vol. 14, Jun. 2023, Art. no. e00436, doi: 10.1016/j.ohx.2023.e00436.

[20] P. Blumenfeld, T. Rogat, and J. Krajcik, "Motivation and cognitive engagement in learning environments," in *The Cambridge Handbook of the Learning Sciences*. Cambridge, U.K.: Cambridge Univ. Press, Jan. 2006, pp. 475–488.

[21] M. Cooney, C. Yang, A. Padi Siva, S. Arunesh, and J. David, "Teaching robotics with robot operating system (ROS): A behavior model perspective," in *Proc. Workshop Teach. Robot. ROS*, 2018, pp. 59–68.

[22] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A Moveit! case study," Apr. 2014, *arXiv:1404.3785*.

[23] N. Correll, R. Wing, and D. Coleman, "A one-year introductory robotics curriculum for computer science upperclassmen," *IEEE Trans. Educ.*, vol. 56, no. 1, pp. 54–60, Feb. 2013, doi: 10.1109/TE.2012.2220774.

[24] O. Kurniawan, N. T. S. Lee, S. Datta, N. Sockalingam, and P. K. Leong, "Effectiveness of physical robot versus robot simulator in teaching introductory programming," in *Proc. IEEE Int. Conf. Teach., Assessment, Learn. Eng.*, 2018, pp. 486–493, doi: 10.1109/TALE.2018.8615190.

[25] C.-C. Wu, I.-C. Tseng, and S.-L. Huang, "Visualization of program behaviors: Physical robots versus robot simulators," in *Informatics Education—Supporting Computational Thinking*, R. T. Mittermeir and M. M. Sysło, Eds., Berlin, Germany: Springer, 2008, pp. 53–62.

[26] M. Myllymäki and I. Hakala, "Distance learning with hands-on exercises: Physical device vs. simulator," in *Proc. IEEE Front. Educ. Conf.*, 2022, pp. 1–6, doi: 10.1109/FIE56618.2022.9962747.

[27] O. Kurniawan, N. T. S. Lee, and N. Sockalingam, "Is augmented reality robot as effective as physical robot in motivating students to learn programming?," in *Proc. IEEE Int. Conf. Eng., Technol. Educ.*, 2021, pp. 1–8, doi: 10.1109/TALE52509.2021.9678820.

[28] S. Balakirsky, S. Carpin, G. Dimitoglou, and B. Balaguer, "From simulation to real robots with predictable results: Methods and examples," in *Performance Evaluation and Benchmarking of Intelligent Systems*. Norwell, MA, USA: Springer, 2009.

[29] C. K. Liu and D. Negrut, "The role of physics-based simulators in robotics," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 4, no. 1, pp. 35–58, May 2021, doi: 10.1146/annurev-control-072220-093055.

[30] H. Choi et al., "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward," *Proc. Nat. Acad. Sci.*, vol. 118, no. 1, Jan. 2021, Art. no. e1907856118, doi: 10.1073/pnas.1907856118.

[31] V. Lowell and D. Tagare, "Authentic learning and fidelity in virtual reality learning experiences for self-efficacy and transfer," *Comput. Educ.: X Reality*, vol. 2, Jan. 2023, Art. no. 100017, doi: 10.1016/j.cexr.2023.100017.

[32] C. Bossard, G. Kermarrec, C. Buche, and J. Tisseau, "Transfer of learning in virtual environments: A new challenge?," *Virtual Reality*, vol. 12, pp. 151–161, Sep. 2008, doi: 10.1007/s10055-008-0093-y.

[33] D. N. Perkins and G. Salomon, "Transfer of learning," *Int. Encyclopedia Educ.*, vol. 2, pp. 6452–6457, 1992.

[34] M.-J. Martínez-Argüelles, D. Plana-Erta, and À. Fitó-Bertran, "Impact of using authentic online learning environments on students' perceived employability," *Educ. Technol. Res. Dev.*, vol. 71, no. 2, pp. 605–627, Apr. 2023, doi: 10.1007/s11423-022-10171-3.

[35] S. E. Lakhan and K. Jhunjhunwala, "Open source in education," *Educause Quart.*, vol. 31, no. 2, pp. 32–40, 2008.

[36] "Getting started with KUKA robot learning lab at KIT." Accessed: Dec. 12, 2022. [Online]. Available: https://rll-doc.ipr.kit.edu/getting_started.html

[37] "Gzweb, the WebGL client for Gazebo." Accessed: Jan. 19, 2023. [Online]. Available: https://classic.gazebosim.org/gzweb

[38] M. Schmittle et al., "OpenUAV: A UAV testbed for the CPS and robotics community," in *Proc. IEEE/ACM 9th Int. Conf. Cyber-Phys. Syst.*, 2018, pp. 130–139, doi: 10.1109/ICCPS.2018.00021.

[39] A. M. Potdar, N. D G, S. Kengond, and M. M. Mulla, "Performance evaluation of Docker container and virtual machine," in *Proc. 3rd Int. Conf. Comput. Netw. Commun.*, 2020, vol. 171, pp. 1419–1428, doi: 10.1016/j.procs.2020.04.152.

[40] "TigerVNC, high performance, multi-platform VNC client and server." Accessed: Jan. 7, 2023. [Online]. Available: https://github.com/TigerVNC/tigervnc

[41] "The VirtualGL project, background." Accessed: Jan. 7, 2023. [Online]. Available: https://virtualgl.org/About/Background

[42] "NVIDIA CUDA GL, extends the CUDA images by adding support for OpenGL through libglvnd." Accessed: Jan. 7, 2023. [Online]. Available: https://hub.docker.com/r/nvidia/cudagl

[43] "NVIDIA cloud native documentation." Accessed: Jan. 7, 2023. [Online]. Available: https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html

[44] J. Martin, S. Mannehed, and P. Ossman, "noVNC - the open source VNC JavaScript client." Accessed: Jan. 7, 2023. [Online]. Available: https://novnc.com/info.html

[45] "Websockify, WebSocket to TCP proxy/bridge." Accessed: Jan. 20, 2023. [Online]. Available: https://github.com/novnc/websockify

[46] P. Dias, "Dockerode, a node.Js docker remote API module." Accessed: Jan. 20, 2023. [Online]. Available: https://github.com/apocas/dockerode

[47] "Open robotics, the official ROS wiki, ROS technical overview." Accessed: Jan. 20, 2023. [Online]. Available: https://wiki.ros.org/ROS/Technical%20Overview

[48] W. Reese, "NGINX: The high-performance web server and reverse proxy," *Linux J.*, vol. 2008, no. 173, 2008, Art. no. 173.

[49] L. Joseph and A. Johny, "Getting started with Ubuntu Linux for robotics," in *Robot Operating System (ROS) for Absolute Beginners: Robotics Programming Made Easy*, L. Joseph and A. Johny, Eds., Berkeley, CA, USA: Apress, 2022, pp. 1–52.

[50] "Picking the right operating system for enterprise robotics," [White paper], Canonical, Apr. 2022. Accessed: Nov. 7, 2023. [Online]. Available: https://ubuntu.com/engage/robot-operating-system-choice

[51] S. Schumann, D. Krūmiņš, V. Vunder, A. Aabloo, L. A. Siiman, and K. Kruusamäe, "A beginner-level MOOC on ROS robotics leveraging a remote web lab for programming physical robots," in *Robotics in Education*, R. Balogh, D. Obdržálek, and E. Christoforou, Eds., Cham, Switzerland: Springer, 2023, pp. 285–297.

[52] M. Promonet, "WebRTC streamer for V4L2 capture devices, RTSP sources and screen capture," release v0.7.0. [Online]. Available: https://github.com/mpromonet/webrtc-streamer/releases/tag/v0.7.0''

[53] J. Brooke, "SUS: A "quick and dirty" usability scale," in *Usability Evaluation in Industry*. Boca Raton, FL, USA: CRC, 1996.

[54] J. Goopio and C. Cheung, "The MOOC dropout phenomenon and retention strategies," *J. Teach. Travel Tourism*, vol. 21, no. 2, pp. 177–197, Apr. 2021, doi: 10.1080/15313220.2020.1809050.

[55] S. I. de Freitas, J. Morgan, and D. Gibson, "Will MOOCs transform learning and teaching in higher education? Engagement and course retention in online learning provision," *Brit. J. Educ. Technol.*, vol. 46, no. 3, pp. 455–471, May 2015, doi: 10.1111/bjet.12268.

[56] A. Bangor, P. Kortum, and J. Miller, "Determining what individual SUS scores mean: Adding an adjective rating scale," *J. Usability Stud.*, vol. 4, no. 3, pp. 114–123, May 2009.

[57] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues, R. L. F. Cunha, and R. Buyya, "HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges," *ACM Comput. Surv.*, vol. 51, no. 1, 2018, Art. no. 8, doi: 10.1145/3150224.

[58] P. Prabhu et al., "A survey of the practice of computational science," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage, Anal.*, 2011, pp. 1–12, doi: 10.1145/2063348.2063374.

[59] A. Thota et al., "Research Computing desktops: Demystifying research computing for non-Linux users," in *Proc. Pract. Exp. Adv. Res. Comput. Rise Mach.*, 2019, pp. 1–8, doi: 10.1145/3332186.3332206.

[60] V. Mittal, L.-H. Hung, J. Keswani, D. Kristiyanto, S. B. Lee, and K. Y. Yeung, "GUIdock-VNC: Using a graphical desktop sharing system to provide a browser-based interface for containerized software," *GigaScience*, vol. 6, no. 4, Apr. 2017, Art. no. giw013, doi: 10.1093/gigascience/giw013.

[61] T. Combe, A. Martin, and R. Pietro, "To Docker or not to Docker: A security perspective," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 54–62, Sep./Oct. 2016, doi: 10.1109/MCC.2016.100.