Robotics Education and Research at Scale: A Remotely Accessible Robotics Development Platform

Wolfgang Wiedmeyer[1], Michael Mende[1], Dennis Hartmann[1], Rainer Bischoff[2], Christoph Ledermann[1] and Torsten Kröger[1]

***Abstract*— This paper introduces the *KUKA Robot Learning Lab at KIT* – a remotely accessible robotics testbed. The motivation behind the laboratory is to make state-of-the-art industrial lightweight robots more accessible for education and research. Such expensive hardware is usually not available to students or less privileged researchers to conduct experiments. This paper describes the design and operation of the Robot Learning Lab and discusses the challenges that one faces when making experimental robot cells remotely accessible. Especially safety and security must be ensured, while giving users as much freedom as possible when developing programs to control the robots. A fully automated and efficient processing pipeline for experiments makes the lab suitable for a large amount of users and allows a high usage rate of the robots.**

I. Introduction

Despite continuously decreasing costs, robots are still expensive hardware. This is especially true for state-of-the-art industrial lightweight robots with force/torque sensing capabilities in every joint. In order to be able to use the robots for various applications, research facilities and universities set up general-purpose robot cells with additional sensors. Maintaining these experimental robot cells requires experience and time. Furthermore, software interfaces are usually developed with a specific application in mind which causes an additional overhead when repurposing the robot cell for different types of experiments. Safety measures such as workspace restrictions and collision detection algorithms are often not sufficient which requires constant monitoring of experiments to avoid damages to the hardware.

For these reasons, it is common that only select students are able to use such robot cells within the scope of long-term projects like theses. Most need to resort to simulation, in the same way as researchers without sufficient funding. However, current simulation frameworks do not sufficiently reflect real-world behavior, especially when manipulating objects or emulating sensors. A simulation-only approach prevents researchers from validating their methods with real-world experiments. From a student's perspective, exploring the foundations of robotics by being able to control a real robot that interacts with the real world can be a big motivational factor in education and allows to gain experience with robotic hardware. The above issues are addressed by the Robot Learning Lab.

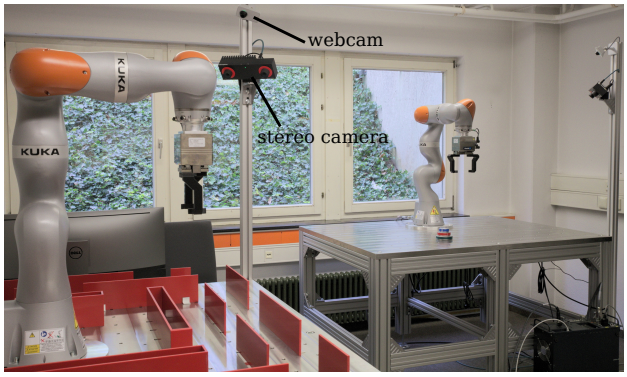

Fig. 1. The two robot cells of the Robot Learning Lab

The Robot Learning Lab, as shown in Figure 1, is a remote-access robotics testbed, explicitly designed to make robotic manipulators more available for students and researchers by providing access over the Internet to robotic work cells. These cells are flexible, so they allow for various different experimental setups, i.e. for educational purposes or forefront research challenges. What sets the Robot Learning Lab apart from other remotely accessible testbeds is its focus on supporting the development of applications with arbitrary complexity and without restrictions for research or educational purposes, while maximizing scalability through automation and minimal invasive safety/security measures. Other testbeds lack flexibility or the ability to write code in general-purpose programming languages to be considered development platforms, or they are not able to handle a large group of users and multiple robot work cells because experiments are not performed fully automated and high availability of the robots in the lab is not guaranteed. In this paper, we discuss how the Robot Learning Lab is structured and, in particular, how it constitutes a *flexible*, *safe*, *secure* and *scalable* remote-access development platform.

II. Related Work

In this section, we briefly give an overview of remote access testbeds that have been successful in their respec-

[1]Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Germany, {wolfgang.wiedmeyer, michael.mende, dennis.hartmann, christoph.ledermann, torsten}@kit.edu.
[2]KUKA Deutschland GmbH, Augsburg, Germany, rainer.bischoff@kuka.com

tive domains and categorize them according to the robotic hardware they employ: stationary robots and mobile robots. The following survey does not include remotely accessible robotics labs that only provide teleoperation or telepresence capabilities as these are not suitable for software development.

A. Stationary Robot Testbeds

A few remote robotics labs have been developed that use a stationary mounted robotic manipulator, like the industrial robots in the Robot Learning Lab. The Telerobot [1] and RACT lab [2] each employ an industrial robot, while [3], [4], [5], [6] use an educational robotic manipulator. [1], [3] allow students to program the robot for pick-and-place tasks with the help of a graphical user interface. [4] also supports the upload of code files generated from a simulator. A visual servoing task is additionally provided by [2]. [5] also gives the ability to use robot-specific code-like instructions through the web interface.

While these remote robotics labs were pioneering work at their time, they lack the flexibility to conduct research experiments. In contrast to the Robot Learning Lab, they do not provide the ability to submit programs in a general-purpose programming language and the feature set is almost entirely limited to what is needed to teleoperate the robots in the lab. In comparison, the Robot Learning Lab is designed to host projects with arbitrary complexity, either for educational or research purposes, and to serve as a development platform. With the exception of [6], only basic safety needs are addressed for the specific projects that were developed for the labs. Safety is not treated in a general and holistic fashion and the authors do not present safety concepts that are applicable for different experimental setups.

B. Mobile Robot Testbeds

In recent years, the development of remote robotics labs with stationary robots ceased and mobile robot testbeds became popular, especially with multiple robots. The PR2 Remote Lab [7] gives access to a PR2 personal robot, the RPN provides a NAO humanoid for remote access and the Robotarium [8], SyRoTek [9] and Robotnačka [10] deploy small wheeled robots. The PR2 Remote Lab [7] and the Robotarium [8] have a research and development focus, while the RPN [11], SyRoTek [9] and Robotnačka [10] target online education. Out of these, only the Robotarium has safety measures that prevent damages to the robots, even from untrustworthy or malicious users. However, the safety challenges for swarm robotics significantly differ from the ones for industrial robots. In contrast to the above described testbeds with stationary robots, all of the mobile robot labs allow local execution of submitted user code, but security risks of code compromising local machines are not sufficiently or not at all addressed, especially in regard to network security.

A few sensor networks testbeds exist that also provide mobile robots, most notably the FIT IoT-LAB [12]. But their research focuses on networking solutions in an IoT context and does not target robotics.

For no remote access lab, neither with stationary robots nor with mobile robots, the authors describe methods for having a queue of submitted experiments and to perform experiments fully automated for several users, especially in parallel for multiple robot work cells, which would be required to accommodate a large number of users. In fact, none of the described labs has multiple work cells like the Robot Learning Lab.

III. The Robot Learning Lab

In this section, we present the design considerations that guided the development of the Robot Learning Lab, and elaborate on both the hardware and software architectures of the Robot Learning Lab.

A. General Design Considerations

As a remotely accessible robotics development facility, the Robot Learning Lab's main purpose is to increase the availability of state-of-the-art industrial robots for education and research. Thus it has to implement a number of high-level design requirements aimed at accessibility, scalability, automation and safe and secure code execution.

- Provide enough flexibility to host research projects and challenging robotics competitions, while also being able to teach students the foundations of robotics with a low barrier of entrance.
- Enable fully automated execution of code submissions with an efficient job processing pipeline that can handle a large amount of users and allows for parallel execution of jobs in multiple robot work cells, and that requires minimal maintenance and monitoring by a human operator.
- Integrate safety and security measures to protect the Robot Learning Lab from damage and misuse through safe operation of the robots with collision checks and user code isolation.
- Allow users to develop applications with the help of a simulation that closely mirrors the setup in the lab, and ensure that these applications can be run in the lab without the need for additional changes.
- Enable interaction with and data retrieval from the lab through a web API that can be integrated into a multitude of services (e.g. online education providers, university courses, competition websites) or client types (e.g. web, mobile, command-line).
- Make it easy to design projects with their own experimental setups in the robot cell and software interfaces.

B. Hardware Setup

The lab consists of two robot work cells with the exact same hardware setup, as shown in Figure 1. Each is equipped with a Kuka LBR iiwa 7 R800 lightweight robot. The iiwa is a redundant industrial manipulator with seven joints and force/torque sensing capabilities in every joint [13]. A SCHUNK EGL 90-CN gripper [14] is attached to the

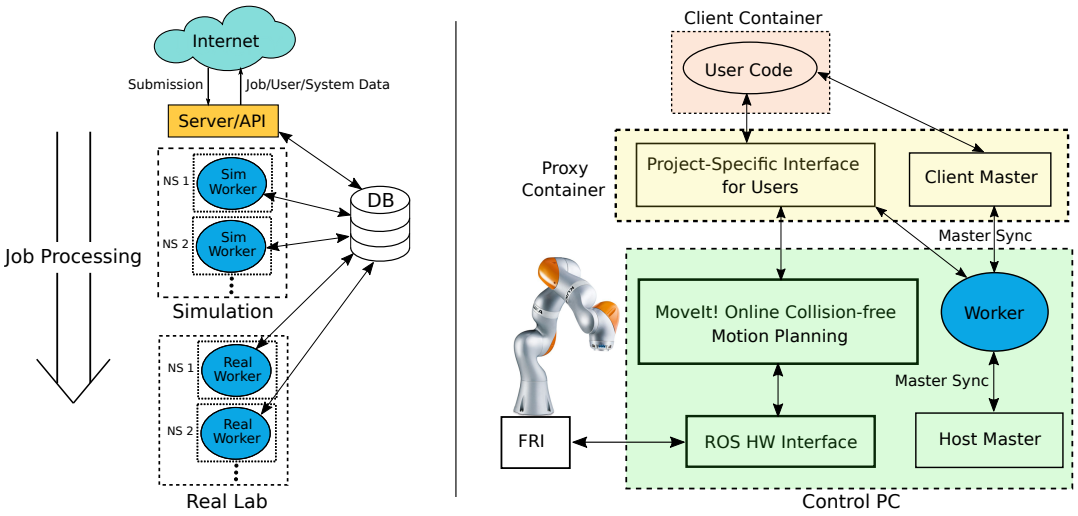

Fig. 2. Overview of the job processing pipeline and the user code execution. The left figure shows the job processing pipeline that includes the web API and the workers for simulation and the real robots. The right figure illustrates how user code is executed by the worker in an isolated environment with on-line checks for collision-free execution on the robot.

robot's end-effector. The robot is mounted to a table with a stainless steel plate. The steel plate's surface measures 120x160 cm and it has a grid with threaded holes in an equidistant spacing of 100 mm to allow for the reliable attachment of different experimental setups. By using an adapter plate and dowel pins, the robot can be mounted in different locations on the table according to the grid and thus exact positioning is guaranteed. With a weight of 500 kg, the steel plate minimizes vibrations of the robot base and ensures that the robot can execute highly dynamic trajectories with high precision.

A Roboception rc_visard 160 stereo camera [15] is mounted to an aluminum profile at the side of the table. It allows for visual monitoring of the work cell and the point cloud data from the camera can be used in experiments. At the top of the profile, a network camera is attached that is used to record or livestream experiments. Besides mounting directly to the steel plate, further hardware extensions to the setup can easily be added by mounting to the support profiles of the table.

C. Software Infrastructure and Design

The software components managing the operation of the Robot Learning Lab can be broadly grouped into three categories: applications for the processing of submissions and user interaction with the lab, components related to safety and security, and components that facilitate application development and project design.

1) Submission Processing and Interaction with the Lab: The Robot Learning Lab is able to fully automatically process a submission. By making a new submission through the web interface, a user triggers the scheduling of a single execution of his submitted code. In the following, such a submission is called a "job". Each job is tracked in the system by a unique job ID. Jobs need to be submitted for a specific "project" and the submitted code, that corresponds to a job, should provide solutions to the tasks that are defined within a project. A task could be to implement an inverse kinematics solution for the robot or to detect the poses of objects that should be grasped by the robot. A project is defined by a certain experimental setup in a robot work cell and by its own software user interfaces that are supplied by a project software package. The system is aware for which projects the robot work cells are currently configured and processes the jobs accordingly. The jobs are processed by "workers" which will be further detailed in the following (see Figure 2 for an overview).

Outside communication with the Robot Learning Lab is possible through a RESTful web API. Due to its non-blocking network I/O features, the server can handle a large amount of open connections and allows large numbers of users to submit experiments and retrieve data in parallel. It is possible to get the current status of a job and data for the job after it has finished. Jobs are queried by their ID which is provided to the user when the job is successfully submitted. For finished jobs, log files, a video recording of the job and optionally further data collected during the job run is provided.

The status of the job is reflected by several status codes like "downloading code", "building", "finished". When a job is submitted and waiting for execution, the position in the queue is returned. If the job is currently running in the lab, a link to the live video feed of the network camera in the respective robot work cell, where the job is executed, is returned. A new submission can be made by providing an archive of the code[1].

A job is first executed in simulation. The simulation environment is the same one that is provided to users (see Section III-C.3). The main purpose of the simulation run is

[1]The full documentation of the web API is available at `https://rll-apidoc.ipr.kit.edu/`. The documentation website is interactive and requests can directly be tested on the webpage.

to check if the user code can be executed by the pipeline. Section III-C.2 gives an overview of the environment in which the user code is executed. The simulation check provides users with early feedback if they did any mistakes that could prevent their code from running in the lab, and it ensures that usage time of the robots is not wasted by erroneous user code.

Simulation runs are controlled by sim workers and code executions for the robot work cells in the lab are handled by real workers (see Figure 2). The expressions "sim" and "real" are generally used to differentiate between simulation and the real robot work cells in the lab. The workers are responsible for code download, building the user code and code execution. They also retrieve and save the job data, including job metadata like job result codes (e.g. "build failed", "sim success", "launching project failed"), and trigger a reset of the robot work cell back to the start state after a job run. The server and workers communicate through a database that stores the job metadata. The workers are configured to process jobs for a specific project. They search the database for jobs, that were submitted for this specific project, and select the oldest submission as the next job to be executed.

The Robot Learning Lab leverages the ROS robotics middleware. Every worker instance is isolated in its own ROS namespace, together with the user code execution environment. This way, several sim workers for parallel simulation checks and multiple real workers for an arbitrary number of robot work cells can run at the same time and process jobs in parallel.

The worker and the server as part of the job processing pipeline are hardware-agnostic and can be used with any robotic hardware. Alongside this paper, we are publishing their source code as free and open-source software in the hope, that it will allow others to build their own remotely accessible robotics testbed.[2]

2) Safety and Security: The system is able to detect failures that cannot automatically be corrected by itself. Such "internal errors" are registered during job execution when methods fail that should always succeed or when the system detects an unexpected or unknown state of the environment (simulation or real) from which it cannot get back to a desired state. Internal errors are also reported if the environment reset failed after job execution. Although the system and project design should minimize the likelihood of internal errors, users might still be able to trigger them. If a worker detects an internal error, it immediately stops its job execution environment including the user code, notifies the operator and terminates itself so that no further jobs are executed in the environment. This allows an operator to inspect the error and ensures that the system does not operate in an undesired state which can lead to hardware damage.

For collision-free execution of robot trajectories, the motion planning framework MoveIt! [16] is used. MoveIt! is able to construct a collision model for the robot cell from the environment description that is defined for every project[3]. We expanded MoveIt! to make it possible to parameterize cartesian trajectories with a fixed cartesian speed. To reduce jerk of point-to-point trajectories, we use the TOTG algorithm introduced in [17].

The iiwa robot provides the Fast Robot Interface (FRI) which allows for real-time control of the robot via a network interface. We implemented a ROS hardware interface for position control using the FRI API in order to use FRI together with MoveIt! (see Figure 2).

For security reasons, the user code is compiled and run inside a Docker container without root privileges. Docker containers are more lightweight than virtual machines because they directly run on the host system's kernel. It is possible to restrict system resources for containers, like the number of CPU cores, RAM and disk space. For every project, an image with all the needed software packages and configurations can be created and the worker sets up a proxy and client container based on the images and additional configuration options. The client container runs the downloaded user code and the proxy container provides the project-specific interface for the user code and runs the separate ROS master for the client. The worker places the downloaded user code in the client container and extracts log files from both the proxy and client container.

The main purpose of the proxy container is to prevent the user code from accessing the host network and the lab network. If users were able to call arbitrary ROS network resources or even send commands directly to the robot, they could bypass all safety measures and cause serious damage to the robot work cells. For every job, the worker starts a new client container and attaches it to the same Docker network as the one the proxy container uses. The client container only has access to the proxy container, while the proxy container is also able to connect to the lab network. The project-specific interface inside the proxy container offers all the ROS network resources the user needs for the project (see Section III-C.3).

The separate ROS master inside the proxy container allows the user code to look up the IP address and ports of interface nodes and to register its own services, topics, actions and parameters. The interface in the proxy container uses the ROS master from the host, so that ROS nodes running on the host are able to connect to it. The worker needs to synchronize the registrations that the interface makes at the host master to the client master so that the user code is able to see them.The registrations at the client master by the client code are synchronized selectively by the worker to the host master, so that the interface can communicate with client implemented code (see Figure 2). The worker reads the registrations that need to be synchronized from project-specific configuration files.

The custom master synchronization implementation for the worker was chosen over existing multi-master solutions

[2]The pipeline's source code can be found at `https://github.com/KITrobotics/rll_stack`.

[3]The robot cell is modeled with the Unified Robot Description Format (URDF), see `http://wiki.ros.org/urdf/XML`

because they target discovery and synchronization of ROS systems on unreliable networks and lack the flexibility to synchronize selected registrations on an on-demand basis. Multiple ROS security extensions have been proposed [18], [19], [20] and ROS 2.0 includes similar security features [21]. However, the aim of these security enhancements is to allow for fine-grained access control and transport encryption which would require a significant configuration overhead for every project, and users would not be able to add their own ROS network resources. The user code isolation in its own Docker container and via a proxy container separates the user code from the ROS system of the lab and still allows to relay any ROS host resource to the client container. Even if the user code manages to crash the interface or the client master in the proxy container, the safety of the system is still ensured.

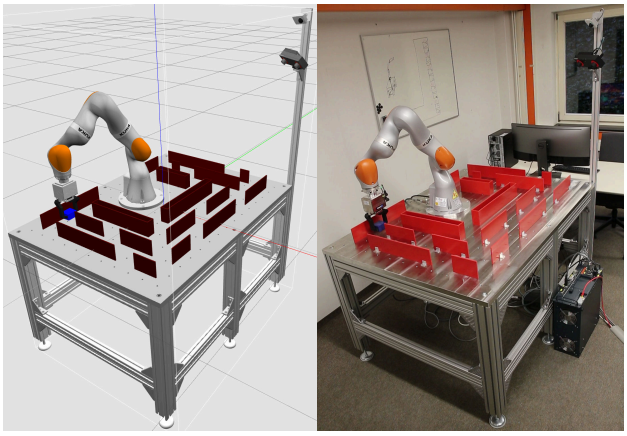

Fig. 3. The simulation and the real robot cell side-by-side

3) Application Development and Project Design: Users can write their own applications for a specific project that is hosted in the Robot Learning Lab. Besides letting users participate in projects, we also want to make it straightforward for interested parties to design their own projects for the Robot Learning Lab. A project design can have its own experimental setup in a robot work cell and custom user-facing interfaces can be implemented.

Applications can be written in C/C++, Python and Bash[4]. The complete ROS package set is available to users and further libraries can be added to the Docker container images if needed. Gazebo[5] is used for the simulator and an exact model of the robot work cell can be used for development (see Figure 3). Thanks to Gazebo's ROS integration, it is possible to replace the hardware interface for the real robot (see Section III-C.2) with a control layer for the simulation. This way, identical user-facing interfaces for controlling the simulated and the real robot can be provided.

Besides a video, users can retrieve the build log, the log file that their code generated during execution and the part of the log file from the interface that corresponds to their job run, both for the simulation and the real run. These files usually suffice for debugging and evaluation. Furthermore, data can be recorded during job execution and provided as ROS bag files[6] to users.

The software structure of every project needs to consist of two parts: user-facing interface nodes and the user code. While users can run both on the same PC they use for development, the interface nodes are meant to be running in the proxy container and the user code should run in the client container in the Robot Learning Lab (see Section III-C.2). The user code is developed by the users of the Robot Learning Lab and it accesses ROS resources provided by the interface nodes. The interface nodes can implement new functionality or they directly relay other resources like sensor data.

For every project, it is expected that the user-facing interface has routines to start and stop a job and to reset the environment to a start state. The environment reset should not only move the robot back to its home position, but the robot also needs to place all objects that can be manipulated back to their initial pose. It should also offer some way to control the robot. To ease the development of interfaces and facilitate the project designs for the Robot Learning Lab, we are publishing a SDK that contains the simulation environment and a library for interface development[7]. The library abstracts MoveIt! functionality and directly works with the simulation environment. For controlling the real robot, a slightly modified library with driver abstractions and improved trajectory generation but with the same API is used, so that no changes are required to the project-specific interface code. The library included in the SDK defines a set of services that allow users to send target poses in joint space or cartesian space. For target poses in cartesian space, linear or point-to-point trajectories can be used. A service for picking up or placing a grasp object is available as well.

IV. Usage

With three examples, this section highlights the main usage features of the Robot Learning Lab: safe and secure remote access for application development and automated continuous operation.

A. Udacity Intersect Conference

The Robot Learning Lab was presented at the Udacity Intersect conference in Mountain View, California, USA. Visitors were able to choose between two pre-programmed projects on a website and trigger remote execution of the selected project in the Robot Learning Lab. The website used the Robot Learning Lab API (see Section III-C.1) as backend and provided conference participants with status information of their submitted job, a video live stream of the job and the log file of the project code. Project submissions

[4]Support for further programming languages can be added as long as a ROS client library for this language exists. Available client libraries are listed at `http://wiki.ros.org/Client%20Libraries`.

[5]`http://gazebosim.org/`

[6]`http://wiki.ros.org/Bags`

[7]The Robot Learning Lab SDK is available at `https://github.com/KITrobotics/rll_sdk`

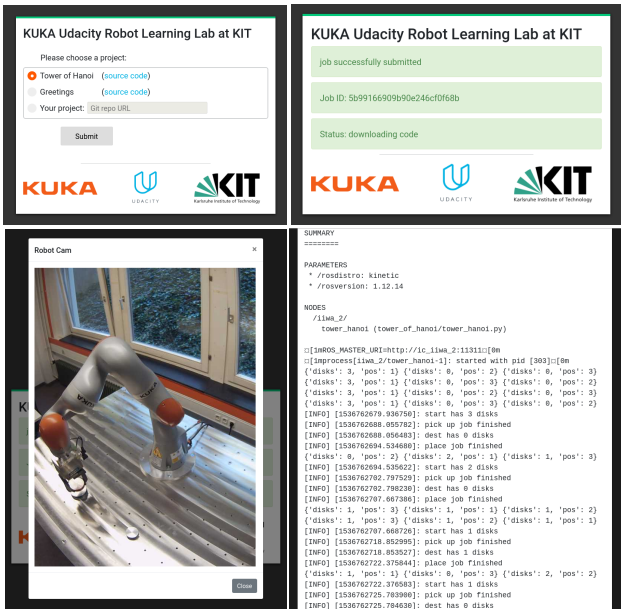

Fig. 4. Webpage for the Udacity Intersect conference. Participants of the conference were able to select one out of two demo projects and view status information, a video livestream and the user log file of their job.

were distributed among the two robot work cells by the job processing pipeline.

One of the two project options lets the robot execute a waving movement to greet visitors through the webcam video feed. The other one makes the robot play the Tower of Hanoi game with three disks[8]. Both projects use the default robot move interface from the Robot Learning Lab SDK (see Section III-C.3). This initial use case of the lab proved that the job processing pipeline works robustly and the jobs are successfully distributed among the robots. However, the usage rate was low and both robot work cells had the same experimental setup.

B. Path Planning Challenge

A competition for students of the Udacity Robotics Software Engineer Nanodegree program was carried out. Within this competition, students should implement a program to navigate a grasped object through a maze to a goal pose as fast as possible. Through a custom user interface, students were able to command 2D poses of the robot's end-effector and they could check if a linear path between two end-effector poses is collision-free. Navigation around corners in the maze is not possible without rotating the end-effector, which required students to solve a 3D path planning problem. Students were free to implement any path planning algorithm, but most students chose an A*- or a RRT-based solution.

The path planning project was designed using the Robot Learning Lab SDK[9] and the Robot Learning Lab API was integrated into the Udacity online student workspace. Over the course of one month, students were able to submit their code and retrieve data of their jobs[10]. For different start and goal poses, leaderboards in the online workspace ranked submissions according to the time the planning algorithms needed to transport the grasp object from start to goal.

C. Long-term Operation

To demonstrate the ability to continuously operate the Robot Learning Lab for many users, we operated the lab for 274 minutes[11]. Throughout the experiment, we ran scripts that submitted a selection of solutions for the path planning challenge (see Section IV-B) and the Tower of Hanoi and Greetings projects (see Section IV-A) to the Robot Learning Lab under different usernames. The Tower of Hanoi and Greetings projects were executed in one robot cell, while the different path planning project solutions were run at the same time in the other robot cell with the maze set up. The scripts made more submission than the lab could process right away, so submissions needed to be queued up. This way, the experiment simulated a high usage rate of the lab with different work cell configurations.

V. Conclusion and Future Work

In this paper, we have detailed the development of a remotely accessible robotics development platform for education and research – the *KUKA Robot Learning Lab at KIT*. Beyond introducing the hardware and software components required to enable remote access to experimental industrial robot work cells, the Robot Learning Lab addresses the four key concerns of flexibility, safety, security and scalability. Unlike other remotely accessible testbeds, the Robot Learning Lab has an automated job processing pipeline with the capability to run experiments in parallel for a large group of users. Arbitrarily complex project implementations can be submitted in general-purpose programming languages, while safe and secure execution in the lab is ensured so that the hardware cannot be damaged and the lab system cannot be compromised by malicious users. To demonstrate the flexibility and reliability of this testbed, we have shown two use case examples with external users and conducted a long-term operation experiment.

This paper presents the initial launch of the Robot Learning Lab. We are planning to continue the operation of the lab and to host further projects for students and researchers. Special attention will be devoted to the move interface for controlling the robots in the lab as it lacks the ability to generate highly dynamic motions with real-time constraints and within short control cycles. Force/torque control capabilities are missing as well. The lab will be moved to a new

[8]The source code for both projects is available at the following links: `https://gitlab.ipr.kit.edu/rll/tower_of_hanoi`, `https://gitlab.ipr.kit.edu/rll/greetings`

[9]The source code of the path planning project can be retrieved at `https://github.com/KITrobotics/rll_path_planning_project`

[10]A video showing a selection of six job executions, that were submitted by six different participants, can be found at `https://rll.ipr.kit.edu/videos/path_planning.mp4`

[11]A time-lapse video of the experiment can be viewed at `https://rll.ipr.kit.edu/videos/timelapse.mp4`.

lab space that was specifically designed for it and it will be extended with more robot work cells.

ACKNOWLEDGEMENTS

Special thanks go to Kuka AG, Roboception GmbH and Schunk GmbH & Co. KG for providing the hardware for the Robot Learning Lab. We also would like to thank Karim Chamaa, Dana Sheahen, Michael Conway and Anthony Navarro from Udacity, Inc. for their support and help with the Intersect conference presentation and the student path planning challenge.

REFERENCES

[1] J. Trevelyan, "Lessons learned from 10 years experience with remote laboratories," in *International Conference on Engineering Education and Research*, 2004.

[2] M. Casini, F. Chinello, D. Prattichizzo, and A. Vicino, "RACT: a remote lab for robotics experiments," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 8153 – 8158, 2008, 17th IFAC World Congress.

[3] A. Balestrino, A. Caiti, and E. Crisostomi, "From remote experiments to web-based learning objects: An advanced telelaboratory for robotics and control systems," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 12, pp. 4817–4825, Dec 2009.

[4] O. Goldstain, I. Ben-Gal, and Y. Bukchin, "Remote learning for the manipulation and control of robotic cells," *European Journal of Engineering Education*, vol. 32, no. 4, pp. 481–494, 2007.

[5] C. A. Jara, F. A. Candelas, S. T. Puente, and F. Torres, "Hands-on experiences of undergraduate students in automatics and robotics using a virtual and remote laboratory," *Computers & Education*, vol. 57, no. 4, pp. 2451 – 2461, 2011.

[6] R. Šafarič, M. Truntič, D. Hercog, and G. Pačnik, "Control and robotics remote laboratory for engineering education," *International Journal of Online Engineering (iJOE)*, vol. 1, no. 1, 2005.

[7] B. Pitzer, S. Osentoski, G. Jay, C. Crick, and O. C. Jenkins, "PR2 Remote Lab: An environment for remote development and experimentation," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 3200–3205.

[8] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The Robotarium: A remotely accessible swarm robotics research testbed," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1699–1706.

[9] M. Kulich, J. Chudoba, K. Kosnar, T. Krajnik, J. Faigl, and L. Preucil, "SyRoTek–distance teaching of mobile robotics," *IEEE Transactions on Education*, vol. 56, no. 1, pp. 18–23, Feb 2013.

[10] P. Petrovi and R. Balogh, "Deployment of remotely-accessible robotics laboratory," *International Journal of Online Engineering (iJOE)*, vol. 8, no. S2, pp. 31–35, 2012.

[11] G. A. Casañ, E. Cervera, A. A. Moughlbay, J. Alemany, and P. Martinet, "ROS-based online robot programming for remote education and training," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 6101–6106.

[12] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015, pp. 459–464.

[13] KUKA LBR iiwa. [Online]. Available: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa

[14] SCHUNK EGL. [Online]. Available: https://schunk.com/de_en/gripping-systems/series/egl/

[15] Roboception rc_visard. [Online]. Available: https://roboception.com/en/rc_visard-en/

[16] I. A. Sucan and S. Chitta. MoveIt! [Online]. Available: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa

[17] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," *Robotics: Science and Systems VIII*, 2012.

[18] R. White, H. I. Christensen, and M. Quigley, "SROS: securing ROS over the wire, in the graph, and through the kernel," *CoRR*, 2016.

[19] B. Dieber, S. Kacianka, S. Rass, and P. Schartner, "Application-level security for ROS-based applications," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 4477–4482.

[20] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, and P. Schartner, "Security for the Robot Operating System," *Robotics and Autonomous Systems*, vol. 98, pp. 192 – 203, 2017.

[21] V. DiLuoffo, W. R. Michalson, and B. Sunar, "Robot Operating System 2: The need for a holistic security approach to robotic architectures," *International Journal of Advanced Robotic Systems*, vol. 15, no. 3, 2018.