Towards a Technical Infrastructure for a Remote Collaborative Robotics Laboratory

Louis Kobras; Pierre Helbing; Johannes Nau; Marcus Soll; Bernhard Meussen

2025 7th Experiment@ International Conference (exp.at'25), 2025/2026

IEEE Xplore: https://ieeexplore.ieee.org/document/11348504

DOI: https://doi.org/10.1109/exp.at2565440.2025.11348504

Towards a Technical Infrastructure for a Remote Collaborative Robotics Laboratory

Louis Kobras
NORDAKADEMIE gAG Hochschule der Wirtschaft
25337 Elmshorn, GERMANY
louis.kobras@nordakademie.de

Pierre Helbing
Technische Universität Ilmenau
98684 Ilmenau, GERMANY
pierre.helbing@tu-ilmenau.de

Johannes Nau
Technische Universität Ilmenau
98684 Ilmenau, GERMANY
johannes.nau@tu-ilmenau.de

Marcus Soll
NORDAKADEMIE gAG Hochschule der Wirtschaft
25337 Elmshorn, GERMANY
marcus.soll@nordakademie.de

Bernhard Meussen
NORDAKADEMIE gAG Hochschule der Wirtschaft
25337 Elmshorn, GERMANY
bernhard.meussen@nordakademie.de

***Abstract*—To strengthen the engineering education at the NORDAKADEMIE, a new collaborative robot laboratory has been developed. This laboratory should provide local access as well as remote access. This paper describes how the remote access is realised using the CrossLab architecture. For this, a Python client is used which implements Server Oriented Architecture Services provided by packages that were developed as part of the CrossLab project. In addition, the challenges and limitations found during implementation are discussed.**

***Index Terms*—Collaborative Robotics, Remote Laboratories, Universal Robots UR5e, Engineering Education**

I. Introduction

By taking a critical look on the changing work environment through automation and (weak) artificial intelligence, we can assume that there will not be a decrease in available labor force, but instead workers will have to learn new skills [2], [3]. Multiple studies try to capture the new skills needed, e.g. [4], [5].

To meet demand for future development, a good education in engineering and adjacent areas is needed. One important part of (engineering) education are laboratories [6], which provide both important skills [7] as well as improve students' understanding of domain knowledge [8]. One recent trend is the movement towards remote laboratories [9], [10], [11] (and other non-traditional laboratories [12]) instead of physical laboratories with systems like VISIR [13], WebLab-Deusto [14] / LabsLand [15], GOLDi [16],[17] or CrossLab [18]. For an overview of the current technology stack of remote laboratories, see [19].

In future engineering education, competencies in system automation and work with robots as examples of modern industrial machinery will play an important part [20, Tab. 2]. An industrial robot is a machine that can be programmed to move in multiple axes to interact with its environment and fulfill some given task [21]. Collaborative Robots are a subcategory of industrial robots. A collaborative robot is defined by featuring special safety measures such as force and speed control or safety-rated monitored stops [21, 5.10]. Whereas regular industrial robots may pose a danger for humans due to their high speed and force, collaborative robots are designed to be used alongside humans to help directly with mundane tasks, and thus do not require special safety measures such as cages. Instead, during risk assessment, a safe collaborative workspace where robots and humans share work areas is defined. To close the gap between theory and industrial practice, it is important to include real devices, systems, and techniques students can train to operate [22, p. 1].

To improve the knowledge students possess about of industrial automation, the NORDAKADEMIE gAG Hochschule der Wirtschaft is developing a new collaborative robot laboratory. To facilitate flexibility of teachers and learners, the collaborative robotics experiment should be controllable both in person and remotely. The laboratory is embedded into the engineering curriculum as part of a course on production techniques. In addition to hands-on experience, we want to allow students to access the laboratory remotely, both to acquire skills for *Work 4.0* (see [23]) and to be able to access the laboratory even when they do not have means of physically traveling to it.

In this paper we describe the setup of making the laboratory available remotely by implementing a software client against the CrossLab architecture [18]. The focus is on the technical aspects of the laboratory, which is intended to be used in teaching in higher education at a German University of Applied Science. [24] presents a laboratory course in such a setting as an example for the pedagogical implementation.

II. The Laboratory Setup

Our current laboratory consists of two *Universal Robot UR5e* [26] collaborative robots, which are set up in an identical manner (see Fig. 1): Both robots are mounted in a corner of a mobile laboratory work bench of with a size of 1250 mm by 500 mm. We have a number of end effectors available, e.g.

This research was part of the project *Flexibel kombinierbare Cross-Reality Labore in der Hochschullehre: zukunftsfähige Kompetenzentwicklung für ein Lernen und Arbeiten 4.0* (CrossLab) [1], which is funded by the *Stiftung Innovation in der Hochschullehre*, Germany.

Figure 1. Current setup of the UR5e robot. A 2-finger gripper and a wrist camera are mounted as end effectors. The gripper is holding a block, which should be placed on two other blocks on the table to build a tower. The teaching pendant is placed on the table.

a wrist camera, a vacuum gripper, a two-finger gripper, or a screw driver. While these can be interchanged freely at any time, for the moment we plan to use a *Robotiq 2F-85* gripper [27] together with the *Robotiq Wrist Camera* [28].

A CrossLab experiment consists of several lab devices that are connected directly to each other via peer-to-peer-connections over the internet. For this laboratory, the experiment is configured with two laboratory devices: An Experiment Control Panel (ECP) which runs in the browser via a website to interact with the experiment, and a laboratory device which features the robot workstation and peripheral components (Workstation Laboratory Device). Fig. 2 displays a schematic of the general infrastructure. The workstation device of the laboratory consists of three components:

1) the actual robot that is mounted on a work bench and fitted with a gripper that is part of the actual laboratory;
2) a Raspberry Pi 4B that runs a software client written in Python and connects to said robot; and
3) one (or more) USB webcams connected to the Pi to relay the robot behavior back to the user.

The Raspberry Pi connects to the robot using a basic `asyncio` data stream to transmit data and receive a response. The robot exposes a TCP/IP socket on ports 30001-30003 [29] where it accepts commands or scripts written in its programming language *URScript*, a documentation for which is provided by Universal Robots (see [30] for the English documentation). While *URScript* is technically its own language, it features a Python-like syntax and is thus accessible to people with programming experience. The Raspberry Pi sends the scripts it receives from the ECP to the robot. The ECP can be exchanged with any device that implements the required interfaces; in our case it is a website that embeds a form to submit a file and a media player that can play back a video stream. The two laboratory devices are connected using the CrossLab architecture. The Raspberry Pi acts as a front end for the local device by providing a proxy for the robot over which the CrossLab architecture can connect.

III. The Service Architecture

The CrossLab architecture [31], [18] makes use of a microservice design. One of these services is the *device service* which handles the registration and management of atomic laboratory devices. A device may either be a physical piece of laboratory equipment, cloud software, or an edge-instantiable device like a web page. By utilizing a *booking service* [25], a user can book time slots of serval devices to be used later in an experiment. Experiments are handled using the *experiment service*. Based on the experiment configuration the laboratory devices are connected to each other using peer-to-peer connections. For this, each device offers a list of *producer* and *consumer* SOA (*server oriented architecture*) services[1] – a unified interface for data exchange between the devices that are mapped onto each other for an experiment setup. A SOA service is usually directional, i.e., one device features a producer and another device features a consumer which have to be linked[2].

We implemented our client using the Python packages provided by the CrossLab project [32], [33] (as well as related libraries). The SOA services a device offers are managed by a `DeviceHandler` object where they are registered during setup. The `DeviceHandler` also manages the connection to the CrossLab architecture. The code for setting up the `Handler` and connecting it to CrossLab is given in Fig. 3. Note that in lines 26 and 28, some preparatory tasks were omitted for brevity. The contents of an example configuration file are given in the code snippet in Fig. 4. The configuration options are separated into three categories: robot connection, connection to the CrossLab infrastructure, and logging configuration. Also omitted from the setup code in Fig. 3 (lines 6 and 9) were the setup processes of the two SOA services that are used for this setup which will be discussed separately in the following subsections.

[1]Unfortunately, both the server components as well as the offered communication channels between devices is called *services* in the architecture. In this paper, we use "service" to refer to the server components and "SOA service" to refer to the device communication.

[2]Services can also be a producer and a consumer – thus titled *prosumer* – at the same time. As of February 2025, the only prosumer implemented is the `ElectricalConnection` SOA service, as it opens up a set of pins instead of a singular data handler, thus `ElectricalConnection` SOA services are bi-directional.

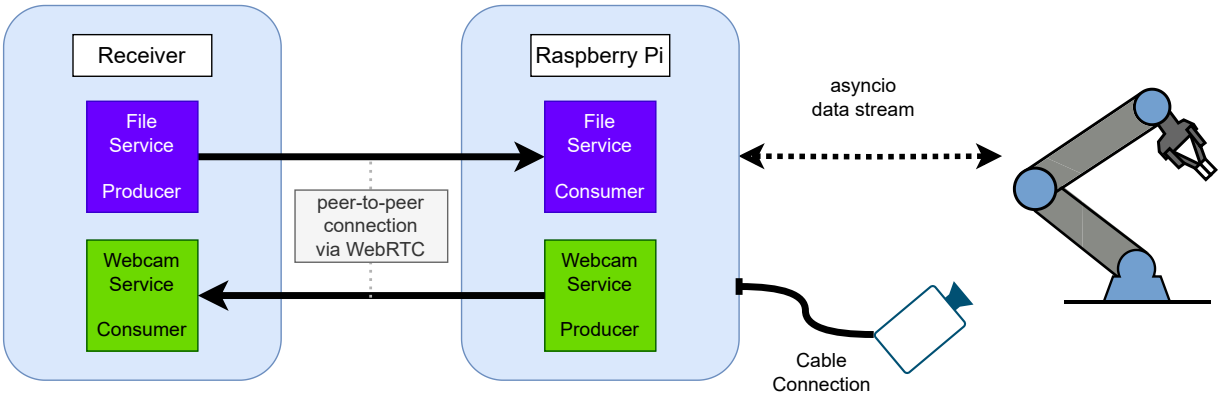

Figure 2. Schematic setup of the laboratory devices. The robot and camera is connected to a Raspberry Pi functioning as a proxy to the CrossLab architecture [18]. Two services are offered: The robot can be controlled via receiving program files and a camera stream is sent. The receiver of both services can either be an end user or another (laboratory) device.

```python
async def main_async():
  # DeviceHandler is the main class to handle the device
  device = DeviceHandler()

  # Snippet to handle the file service
  ...

  # Snippet to handle the webcam service
  ...

  # Authentication and starting the device handler task:
  async with APIClient(config["auth"]["api"]) as apiClient:
    logger.debug("Starting client connection")
    apiClient.authToken = config["auth"]["token"]
    deviceHandlerTask = asyncio.create_task(
      device.connect(
        config["auth"]["api"] + config["auth"]["path"] + config["auth"]["device_id"],
        apiClient
      )
    )
    logger.info("Device set up and ready.")
    await deviceHandlerTask

def main():
  # read config
  ...
  # prepare logging
  ...
  asyncio.run(main_async())


if __name__ == "__main__":
  main()
```

Figure 3. Main program structure. After some initial preparations (reading the config file and setting up a logger) an asynchronous method is started. Within this method, a `DeviceHandler` object is created. A file service (cf. Fig. 5) and a webcam service (cf. Fig. 6) are created and attached to the `DeviceHandler`. The `DeviceHandler` then connects to a CrossLab API server and awaits a PeerConnection.

```json
{
    "robot": {
        "ip": "127.0.0.1",
        "port": 30002
    },
    "auth": {
        "api": "https://crosslab.example.com/",
        "path": "devices/",
        "device_id": "uuid",
        "token": "token-hash"
    },
    "logging": {
        "level": "NOTSET",
        "name": "LOGGER",
        "device": "sysout",
        "filename": "./logfile.log"
    }
}
```

Figure 4. The contents of an exemplary configuration file with dummy values. The configuration is separated in three categories: connection data for the Cobot, authentication data for the API server, and configuration options for the logger. The configuration is formatted in JSON.

A. *The File Service*

This SOA service allows a system to send or receive a small file. We use this service so that a student can submit program code to our robot; however, as other Receivers (cf. Fig. 2) are possible, one might also imagine linking this service to e.g. an AI for auto-generated programs. The submitted program code has to written in *URScript*. The code snippet in Fig. 5 is used to set up a file consumer. The file is received as the content blob of an event. After an event has been triggered, the TCP connection to the robot is opened up using an `asyncio` stream on a configurable IP address and port. Note that it is necessary to wait for the response of the robot: As per documentation, at least 79 bytes should be read from the socket before it closes, otherwise the robot may drop instructions. However, as the read bytes have no defined meaning to a user, we can immediately discard them. Note also the string `'program'` in line 1 – this label will identify

```
fileService = FileService__Consumer('program')
@fileService.on('file')
async def onFile(event: FileServiceEvent):
    logger.debug("onFile")
    # read data from upstream
    data = event['content']
    # write data to robot
    reader, writer = await asyncio.
    open_connection(config["robot"]["ip"],
    config["robot"]["port"])
    writer.write(data)
    await writer.drain()
    logger.debug("Sent program to robot")

    # read robot response
    # undefined and not useful for us; but
    needs to be read as per robot documentation
    _ = await reader.read(100)
    await asyncio.sleep(1) # to wait for the
    robot to be able to read the program
    await writer.wait_closed()
    logger.debug("Reveiced response from robot"
    )
device.add_service(fileService)
logger.debug("File service set up")
```

Figure 5. Code Snippet written in Python for the `FileService Consumer`. A new file service is created and added to the device. When the service receives a file it unpacks the file's contents and transmits them to the robot using `asyncio`. It then reads the response from the robot, but since the content of this response is not useful, it is immediately discarded. Everything from the linked device producing a file to this service receiving it is handled in the background by the CrossLab architecture [18].

the SOA service during the experiment configuration as to be able to differentiate different SOA services of the same type. This service is embedded into the main program (Fig. 3) in line 6.

B. The Webcam Service

This SOA service allows for the transmission of a webcam stream utilizing GStreamer[3] [34] to en- and decode the data stream. The code snippet for setting up the SOA service is displayed in Fig. 6. Constructing the codec pipeline for GStreamer takes up most of this SOA service. The identifying label (in this case `'webcam'`) follows as the last parameter of the constructor. We use this service so that a student can observe the live execution of the program they just submitted. The Raspberry Pi of the local device offers a `WebCam Producer` – a service that transmits a video stream. This stream is caught by the remote device (e.g., the ECP website) and displayed to the user. This service is embedded into the main program (Fig. 3) in line 9.

IV. Trialing the Experiment

To test our implementation of the robot connection to the CrossLab architecture, we manually verified the output of multiple test scripts. These scripts ranged from simple

[3]GStreamer is a multimedia framework for working with different multimedia streams, e.g. capturing a stream from a webcam and saving it in many different formats. It uses plug-ins to support a high number of media formats. It works on all major operating systems.

```
webcamService = WebcamService__Producer(
  GstTrack(
    (" ! ").join([
      "v4l2src device=/dev/video0",
      "'image/jpeg,width=640,height=480,
    framerate=30/1'",
      "v4l2jpegdec",
      "v4l2h264enc",
      "'video/x-h264,level=(string)4'",
    ]),
  ),
  "webcam",
)
device.add_service(webcamService)
logger.debug("Webcam service set up")
```

Figure 6. Code Snippet written in Python for the `WebCam Producer`. A new video stream is created and the service is added to the device. Everything else is handled in the background by the CrossLab architecture [18].

movement containing just a few commands to complex tasks[4]. For example, one script was a *picking* experiment, where the robot has to grip multiple work pieces and build a tower with them. This does not only need high precision (else workpieces would be missed or the tower would fall down), but also the usage of advanced commands (such as stop moving when a reaction force was detected and controlling the gripper). This task is similar to what our students would do and uses capabilities that can be found in industrial settings.

Every script was run locally on the robot and sent through the CrossLab architecture. We were able to verify that all scripts were executed correctly and the output of the robot was identical in both cases for all of our test scripts. Therefore, we were able to conclude that our implementation worked correctly. We plan to use the new interface of the robot in teaching to ensure the implementation works with real student scripts as well as when used under higher load, i.e., when many students send script files in quick succession.

V. Discussion

A. On the Separation of the Client from the Cobot

In Sec. II it was already described that the Cobot is connected to the internet and exposes a socket to receive scripts. If the Cobot already has this capability, why then, one might ask, is the proxy in the form of the Raspberry Pi necessary?

There were two reasons for this decision: firstly, we were unwilling to expose the robot to the public internet. By proxying through a separate device that can only connect directly to the robot in a very specific instance – namely when it receives a script file through an authenticated channel – minimizes the risk of stray login attempts. Of course, allowing one to send a script file which is then executed without any further checks bears risk in itself, especially since the transmitted file contents

[4]In addition, some scripts where created using the teaching pendant and the operating system – *PolyScope* – and exported. Others were programmed manually on a PC. This ensured that different coding styles and formatting worked.

could be anything the connected device wishes to send, script file or not. However the safety mechanisms of the Cobot still apply during script execution, and simple tests[5] resulted in a preliminary finding that a script could not easily break out of its runtime, so ostensibly the worst that could happen would be the Cobot canceling or refusing execution of an invalid program or colliding with something within its working area, in which case it would halt in an emergency state anyway.

The second reason lies with the software. While the Cobot runs on linux and one can connect to it as root using SSH, we consider running software that was not verified by the producer as a bad idea. As we could not estimate how installing third-party packages could mess with the Cobot operating system and its software dependencies and if any legal issues[6] arose – or even if, in the worst case, some of the safety features could get broken this way – we quickly came to the decision that running the CrossLab client on a separate device and using only the provided socket for communication would be the better and more controllable option.

B. Possible Extensions

While providing the ability to run scripts is a good start, we would like to offer more options in the future. Since the CrossLab architecture provides several other SOA services, we would like to take advantage of them. For example, a *Message Service Consumer* could be implemented so users can send singular *URScript* commands to the robot instead of entire scripts. Furthermore, we would like to offer the possibility to users to directly send the TCP (*tool center point)* coordinates through a *Parameter Service Consumer* and let the inverse kinematic of the robot calculate the movement. Finally, we want to streamline using the gripper end effector as part of a program – currently, about 2000 lines of auto-generated boilerplate code need to be copied into a program to have the gripper functionality available.

C. End Effectors

One limitation lies with the availability of end effectors. While the client works fine with a gripper, our Cobot is also equipped with a wrist camera. However, as per inquiry to the manufacturer, it is not possible to use the wrist camera outside of the functionality provided by the teach pendant[7] – this means, in particular, within our remote setup. Thus, while a student working with the physical robot can also utilize the object detection capabilities we have available, remote experimenters are locked out from such experiences. Other end effectors (like a vacuum gripper) might be possible to integrate, but were not tested yet.

D. Limitations

Unfortunately, due to several reasons, we were as of yet unable to test the setup in a live teaching environment. This means that while initial trials were promising, further evaluation is needed, to test the performance at scale, to see how well the setup handles higher request frequencies, as well as to gauge student acceptance and usability.

[5]for example, using the Python function `eval` to try to access system files

[6]e.g., with regards to unpermitted modifications or warranty

[7]As of the time we posed this inquiry (19th Feb. 2024), we used PolyScope 5.12.4.1101661 with the Wristcam plugin on version 1.11.3.13312

VI. Conclusion

This paper presents the technical setup of a client to connect a UR5e collaborative robot with the CrossLab infrastructure. The client accepts a program for the robot as a script file written in *URScript* and provides a webcam stream to observe the robot response. This experiment device, as it is called in CrossLab nomenclature, is intended to train students of engineering and computer science in the programming of industrial machinery. First tests were conducted, although proper tests at scale with many students are still outstanding. Other limitations, as well as possible extensions, were discussed.

While this project certainly has its limitations, we believe it is a step in the right direction. Both remote control of machinery and robotic automation are essential skills in the IoT and Industry 4.0 areas, and as such this project opens up another venue of gaining experience for our students.

References

[1] I. Aubel, S. Zug, A. Dietrich, J. Nau, K. Henke, P. Helbing, D. Streitferdt, C. Terkowsky, K. Boettcher, T. R. Ortelt, M. Schade, N. Kockmann, T. Haertel, U. Wilkesmann, M. Finck, J. Haase, F. Herrmann, L. Kobras, B. Meussen, M. Soll, and D. Versick, "Adaptable digital labs - motivation and vision of the crosslab project," in *2022 IEEE German Education Conference (GeCon)*. Berlin, Germany: IEEE, 2022, pp. 1–6.

[2] L. Willcocks, "Robo-apocalypse cancelled? reframing the automation and future of work debate," *Journal of Information Technology*, vol. 35, no. 4, p. 286–302, 2020. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0268396220925830

[3] L. P. Willcocks, "Automation, digitalization and the future of work: A critical review," *Journal of Electronic Business & Digital Economics*, vol. 3, no. 2, p. 184–199, 2024. [Online]. Available: https://www.emerald.com/insight/content/doi/10.1108/JEBDE-09-2023-0018/full/html

[4] Soll, Marcus and Kobras, Louis and Boettcher, Konrad and Kaufhold, Nils and Schade, Marcel and Terkowsky, Claudius and Helbing, Pierre and Aubel, Ines and Kaiser, Doreen, "Integration of learning outcomes for stem laboratories into a new learning outcome catalogue," in *2025 IEEE Global Engineering Education Conference (EDUCON)*, London, UK, In Print.

[5] M. Winde, J. Klier, V. Meyer-Guckel, E. Schröder, F. Süßenbach, F. Rampelt, D.-K. Mah, S. Buck, S. Hieronimus, J. Kirchherr, M. Keller, M. Metzger, N. Sönmez, F. Schulze Spüntrup, K. Best-Werbunat, and S. Höfer, *Future Skills 2021 - 21 Kompetenzen für eine Welt im Wandel*, Essen, Germany, 2021, no. 3. [Online]. Available: https://www.stifterverband.org/medien/future-skills-2021

[6] B. Jochen and M. Gutmann, "Wozu Labor? Zur Vernachlässigten Erkenntnistheorie Hinter der Labordidaktik," in *Labore in der Hochschullehre: Labordidaktik, Digitalisierung, Organisation*, T. Claudius, D. May, S. Frye, T. Haertel, T. R. Ortelt, S. Heix, and K. Lensing, Eds., 2020, p. 35–49.

[7] N. S. Edward, "The Role of Laboratory Work in Engineering Education: Student and Staff Perceptions," *The International Journal of Electrical Engineering & Education*, vol. 39, no. 1, pp. 11–19, 2002. [Online]. Available: http://journals.sagepub.com/doi/10.7227/IJEEE.39.1.2

[8] F. L. Forcino, "The Importance of a Laboratory Section on Student Learning Outcomes in a University Introductory Earth Science Course," *Journal of Geoscience Education*, vol. 61, no. 2, p. 213–221, 2013. [Online]. Available: https://doi.org/10.5408/12-412.1

[9] J. Ma and J. V. Nickerson, "Hands-on, simulated, and remote laboratories: A comparative literature review," *ACM Computing Surveys*, vol. 38, no. 3, p. 7, 2006. [Online]. Available: https://dl.acm.org/doi/10.1145/1132960.1132961

[10] E. Lindsay, D. Liu, S. Murray, and D. Lowe, "Remote laboratories in engineering education: Trends in students' perceptions," in *Proceedings of the 18th conference of the Australasian Association for Engineering Education*, H. Søndergaard and R. Hadgraft, Eds., Melbourne, Australia, 2007.

[11] J. R. Brinson, "Learning outcome achievement in non-traditional (virtual and remote) versus traditional (hands-on) laboratories: A review of the empirical research," *Computers & Education*, vol. 87, pp. 218–237, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360131515300087

[12] E. K. Faulconer and A. B. Gruss, "A review to weigh the pros and cons of online, remote, and distance science laboratory experiences," *International Review of Research in Open and Distributed Learning*, vol. 19, no. 2, 2018.

[13] F. L. Jacob, M. A. Marques, A. Fidalgo, E. S. C. Ruiz, F. G. Loro, and M. Castro, "Visir remote lab: Identifying limitations and improvement ideas," in *Proceedings of the 20th International Conference on Remote Engineering and Virtual Instrumentation (REV2023)*, 2023, pp. 383–390, in press.

[14] P. Orduña, J. Garcia-Zubia, L. Rodriguez-Gil, I. Angulo, U. Hernandez-Jayo, O. Dziabenko, and D. López-de Ipiña, *The WebLab-Deusto Remote Laboratory Management System Architecture: Achieving Scalability, Interoperability, and Federation of Remote Experimentation*. Cham: Springer International Publishing, 2018, pp. 17–42. [Online]. Available: https://doi.org/10.1007/978-3-319-76935-6_2

[15] P. Orduña, L. Rodriguez-Gil, J. Garcia-Zubia, I. Angulo, U. Hernandez, and E. Azcuenaga, "LabsLand: A sharing economy platform to promote educational remote laboratories maintainability, sustainability and adoption," in *2016 IEEE Frontiers in Education Conference (FIE)*, 2016, pp. 1–6.

[16] K. Henke, T. Vietzke, H.-D. Wuttke, and S. Ostendorff, "GOLDi – Grid of Online Lab Devices Ilmenau," *International Journal of Online and Biomedical Engineering (iJOE)*, vol. 12, no. 04, p. 11–13, 2016. [Online]. Available: https://online-journals.org/index.php/i-joe/article/view/5005

[17] K. Henke, T. Vietzke, R. Hutschenreuter, and H.-D. Wuttke, "The remote lab cloud "goldi-labs.net"," in *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, 2016, pp. 37–42.

[18] J. Nau and M. Soll, "An extendable microservice architecture for remotely coupled online laboratories," in *Open Science in Engineering*, M. E. Auer, R. Langmann, and T. Tsiatsos, Eds. Cham: Springer Nature Switzerland, 2023, pp. 97–109.

[19] M. Soll, J. Haase, P. Helbing, and J. Nau, "What are we missing for effective remote laboratories?" in *2022 IEEE German Education Conference (GeCon)*. Berlin, Germany: IEEE, 2022.

[20] M. Hernandez-de Menendez, C. A. Escobar Díaz, and R. Morales-Menendez, "Engineering education for smart 4.0 technology: a review," *International Journal on Interactive Design and Manufacturing (IJIDeM)*, vol. 14, no. 3, p. 789–803, 2020. [Online]. Available: https://link.springer.com/10.1007/s12008-020-00672-x

[21] ISO 10218-1:2011, "Robots and robotic devices – Safety requirements for industrial robots – Part 1: Robots," International Organization for Standardization, Geneva, CH, Standard, 2011. [Online]. Available: https://www.iso.org/standard/51330.html

[22] C. E. Pereira, S. Paladini, and F. M. Schaf, "Control and automation engineering education: Combining physical, remote and virtual labs," in *International Multi-Conference on Systems, Signals & Devices*, 2012, pp. 1–10.

[23] A. Al-Zoubi, M. Castro, F. R. Shahroury, and E. Sancristobal, "Impact of remote labs in preparing students for Work 4.0," in *2023 IEEE Global Engineering Education Conference (EDUCON)*. Kuwait, Kuwait: IEEE, 2023, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/10125216/

[24] L. Kobras, B. Meussen, and M. Soll, "Didactic design of a remote collaborative robotics laboratory," in *2023 IEEE 2nd German Education Conference (GECon)*. Berlin, Germany: IEEE, Aug. 2023, p. 1–6.

[25] M. Soll, J. Nau, L. Kobras, and P. Helbing, "Building a booking system for federated cross reality laboratories," in *Proceedings of the 22nd International Conference on Smart Technologies & Education (in press)*. Switzerland: Springer Nature, 2025.

WEB LINKS

[26] Universal Robots A/S, "UR5e Lightweight, versatile cobot," https://www.universal-robots.com/products/ur5e/, last accessed 2025-05-06.

[27] Robotiq, "Adaptive Grippers | Robotiq," https://robotiq.com/products/adaptive-grippers#Two-Finger-Gripper, last accessed 2025-05-06.

[28] Robotic, "Wrist Camera | Robotic," https://robotiq.com/products/wrist-camera, last accessed 2025-05-06.

[29] Univesal Robots A/S, "Use the Primary, Secondary, and Realtime Interface to send URScript commands — PolyScope Tutorials documentation," https://docs.universal-robots.com/tutorials/urscript-tutorials/socket-communication.html, last accessed 2025-05-06.

[30] Universal Robots, "Universal Robots - Script Directory - e-Series and UR20/UR30 - SW 5.21," https://www.universal-robots.com/download/manuals-e-seriesur20ur30/script/script-directory-e-series-and-ur20ur30-sw-521/, last accessed 2025-05-06.

[31] jonau, Top-Ranger, and lassertos, "GitHub - Cross-Lab-Project/crosslab," https://github.com/Cross-Lab-Project/crosslab, last accessed 2025-05-06.

[32] jonau, "crosslab-soa-client · PyPI," https://pypi.org/project/crosslab-soa-client/, last accessed 2025-05-06.

[33] ——, "crosslab-api-client · PyPI," https://pypi.org/project/crosslab-api-client/, last accessed 2025-05-06.

[34] "Gstreamer: open source multimedia framework," https://gstreamer.freedesktop.org/, last accessed 2025-05-06.