Education Support Structure for Teaching Multimodal Programming in the Cyber-physical Space

Josef Spillner[a]

Zurich University of Applied Sciences, Winterthur, Switzerland

Abstract: Programming education has become a mandatory element of many engineering curriculums, covering skills from digitally controlled mechanical processes to intelligent traffic and aviation systems. Many of these disciplines require the interaction with physical devices as programming interfaces. Higher education institutions focusing on quality presence labs with the need for occasional online teaching are thus looking for blended and multimodal solutions in which the physical interaction can be carried over as much as possible into the digital channels. In these solutions, various touch points between the physical and digital worlds should be exploited. This paper contributes such a solution. It introduces a cyber-physical educational support structure called EPOSS aimed at programming 'things', including robots and derivative stationary and mobile units, that works in flexible lab and online teaching combinations. The system integrates domain-specific scenarios and open data sources for realistic autoprogramming simulations and is made available as open source prototype to foster adoption. The usefulness of the support system is demonstrated with traffic engineering education scenarios.

1 INTRODUCTION

Programming is widely seen as one of the key competences that any engineer should be able to master (Guo, 2013; dos Santos et al., 2018; Thode et al., 2020). From an economic and employment perspective, this need is driven by the desire to automate processes, to reduce operational errors, and to decompose complex problems into smaller assignable tasks. From a technology perspective, programmability increasingly enters the environments of prospective engineers and of society as a whole. Programming is thus increasingly reflected in teaching to prepare students for the life and work in a software-defined and programmable world. To given an example, an engineering-focused institution like the School of Engineering at the author's higher-education institution requires programming skills across all curriculums:

- Computer science: Students learn foundational concepts of programming and apply them mostly in the digital space (web applications, cloud-native services, machine learning tasks) with some physical touchpoints (computer architectures, sensors).

[a] https://orcid.org/0000-0002-5312-5996

- Business engineering and data science: Students learn with emphasis on automation and apply their skills mostly in the digital space (spreadsheets processing, analytics over unstructured data collections).

- Aviation, traffic systems, environmental sciences, machine technology and electrical engineering: Students are exposed to complex physical systems and have an intrinsic interest to apply their skills in their respective physical domain. This leads to cyber-physical systems (e.g. programmable car with control program) and systems of systems (e.g. transport within one city), with programmable physical 'things' being the smallest unit.

The importance of programming skills was not yet widely acknowledged by most industry domains in the mid-2010s (Prinsley and Baranyai, 2013). Consequently, in the curriculums mentioned last, the acquisition of competences in programming has changed status from being nice to have to being obligatory and highly important. However, this change has happened without the necessary support structures for educators who need to convey characteristic real system behaviour while being able to expose students to these systems directly in most situations. Fail-

ure to convey with the appropriate support leads to a reduction in attractiveness and attention, and eventually to reduced subjective acceptance of programming exercises as integral part of studies (Santana et al., 2018). This problem is reinforced when programming lectures for engineers are merely copying approaches from computer science instead of considering a domain-specific outlook on how the engineers would apply programming in their future careers. Programming real 'things' such as vehicles or production machines could increase attention but requires privileges which are not always attainable in educational settings due to high procurement cost or the sheer size of hardware. Furthermore, students are not always on campus and might be physically separated from both the 'things' and the educators.

This problem has recently become more severe due to forced hybrid and online teaching settings (Ocaña et al., 2020). A lot of the public debate concentrated on collaboration tools for sharing course materials and performing video calls (Klimova and Poulova, 2014), but omitted a critical view on pedagogic concerns specifically for applied learning (Nortvig et al., 2020). Mobile applications and assessment tools for programming education of engineers are available (Ortiz et al., 2015) but likewise do not consider physical objects. The problem rather calls for a substitute environment in which the programming target systems are abstracted to avoid interruptions and upheavals of the respective curriculums. Among the suitable abstractions that can convey a comparable set of characteristics related to the original 'thing' are physical and digital models. The physical models are programmable yet simplified miniature equivalents of the actual 'things', whereas the digital models are objects represented by an application programming interface (API). This leads to the model of Fig. 1 to consider as starting point for the anticipated educational programming and observation support structure (EPOSS).
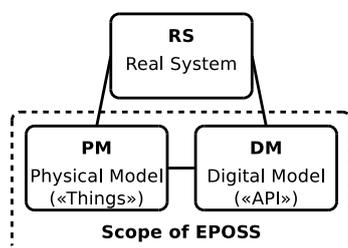

Figure 1: Scope of the support structure among real and model systems.

This paper thus introduces an EPOSS suitable for programming curriculums focused on 'things'. First, it introduces the underlying didactic concepts and functional requirements (Sect. 2). Then, it summarises existing related works aiming at similar concepts (Sect. 3). Afterwards, it dives into the technical realisation. For that matter, it introduces the software system architecture and implementation (Sect. 4) and demonstrates a concrete use case with traffic engineering students (Sect. 5). The paper concludes with remarks on future work (Sect. 6).

2 DIDACTIC CONCEPTS

2.1 Learning Goals and Teaching Concepts

Learning imperative programming, the dominant paradigm for interfacing with the physical world (Smith, 2007), requires learning how to apply algorithmic thinking and planning before proceeding to the implementation. The planning needs to consider all intended effects as well as unintended and unnecessary side effects. In the context of physical objects, the laws of physics and various degrees of imperfection, such as loss of connectivity and unexpected latency, need to be considered in addition to purely digital concerns. For instance, if a mobile robot is instructed to take a turn, this action might take a few seconds during which the submission of further instructions might be blocked. Similarly, if a light sensor shall determine the colour of an object, the lighting conditions in the room and the sensor quality will effect the accuracy (Karaimer and Brown, 2018). Furthermore, real damage might occur if a robot is instructed to move to the cliff of a table in the lab room, and such risks might increase due to limited self-awareness linked to restricted sensing qualities of the hardware available in the classroom. In the orthogonal context of online education, assuming the students do not individually carry the entire physical installation to their places of living, it is important that the consequences of any instructions remain visible and audible immediately.

Hence, the remote teaching of physical objects programming focuses on the following goals:

1. Combination of practical labs and demonstration phases in which the occupancy of the 'things' changes between educators and students.

2. Ability to determine the correctness of a program resulting from practical labs through observation, from both the student's and educator's perspective.

3. Reasoning about the representativeness of physical and digital models for the respective engineering domain.

2.2 Modalities and Student Engagement

Students learn in supervised or unsupervised conditions in on-site, hybrid or online settings with direct or remote access to physical 'things'. Typically, institutions with hundreds of students concurrently enrolled in programming courses will not purchase hundreds of things; rather, they would use coarse-grained or fine-grained time sharing (Meyer and Seawright, 1970) that guarantees exclusive access and, in the case of on-site teaching and coarse-grained time sharing, would be implied by the course schedules. In case students are prevented from participating on-site, there are multiple options - students share things amongst each other, educators parcel things to students, or an online system is used. The first two options are impractical, expensive and, in the case of a pandemic, even impossible, leaving the need for an online support structure.

When no direct access is possible, the unsupervised learning might be constrained. For instance, an educator hosting equipment in the home office could assign time slots for students to use but might not want to be disturbed outside of these slots, given the movement, noise and sensing by programmable 'things'. Furthermore, moving objects typically operate on battery, requiring to be placed on a charger occasionally. While more advanced robots can return for charging autonomously, the budget systems typically available for education require manual placement. Finally, things may crash or get stuck, requiring the manual intervention of the educator.

2.3 Support Structure

To achieve the desired educational quality, a cyber-physical education support system following ideas of virtual labs, but lifting those ideas to the cyber-physical level including virtual/digital and physical elements, shall have the following features:

1. Support for both direct and indirect (distance-proxied) programming of things.

2. Support for partial simulation and human-in-the-loop models to allow for complementary programming and gap filling exercises.

3. Support for data-driven autoprogramming, i.e. the controlled generation of instructions based on external open data sources that represent real systems.

4. Support for observing the effects of the programming or autoprogramming, such as movement of things, through multiple senses - visual, auditive - also across distance if necessary.

5. Safety barriers to avoid damage on things that results from continuous driving against a wall and other non-intentional programming instructions. These barriers are enforced automatically and thus also enable a certain level of self-study without the constant interference of an educator in the presence of mistakes.

The resulting interaction between all structural parts of the education support system, as well as the system interfaces to the student, are shown in Fig. 2.
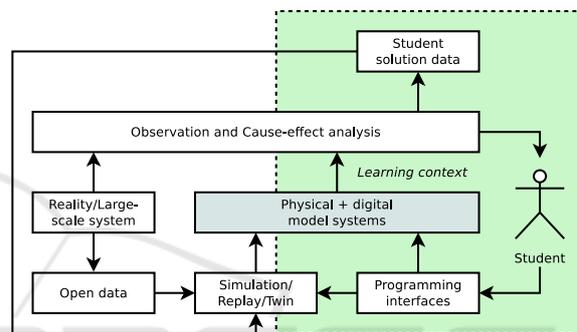

Figure 2: Cyber-physical education support structures from a student perspective.

3 RELATED WORKS

Blending various forms of interacting with physical devices is a recurring topic in education. Simulation, virtual reality (VR) and digital twinning are among the promising concepts. Some educators propose to incorporate simulations for conceptual grounding, for instance in fluid mechanics (Altuger-Genc et al., 2018). Others propose to combine VR and simulation to bypass physical setups while still conveying a realistic environment (Bolano et al., 2020). Despite being helpful for online learning, many educators consider it mandatory to involve actual physical behaviour due to the absence of affordable precise simulations, due to the higher attractiveness of tangible objects for students, and due to better alignment between learning modalities (Hodges et al., 2020). Such a concept was already proposed and implemented previously in Brazil (Almeida et al., 2017) involving multi-agent systems, schedules and integration into a Learning Management System (LMS). One difference to our work is that in that system, programming commands would be necessarily submitted through the LMS, whereas the proposed EPOSS aims at a seamless inte-

gration, not requiring any forms of programming interaction that differs from the physical ones. A further difference is that the study was conducted with school children in the age bracket of 12–17 years, whereas the EPOSS is designed for teaching to undergraduates and graduates with different curricular interests, not necessarily focusing on robots but all relating to programming, without a particular age bracket but generally being adults.

Remote, hybrid and on-site physical programming through virtual laboratories has been proposed for industrial automation in Mexico and Spain (Vázquez-González et al., 2018). It is a promising proposal but lacks a concrete implementation. Another approach is the modelling of cyber-physical system applications in web browsers using high-level abstractions (Peter et al., 2015). This approach was validated with students attending embedded systems courses and led to positive feedback by students. Among these related works, none exploit network transparency to seamlessly combine the various presence and online teaching modes.

4 SYSTEM ARCHITECTURE AND IMPLEMENTATION

4.1 Design and Architecture

EPOSS is designed as a software system to support all processes around physical objects programming education. Its architecture is adapting to the modalities of teaching, including on-site classrooms, hybrid settings with reduced classroom presence, and pure online teaching. Furthermore, it is adapting to the level of guidance from educators, ranging from fully guided labs and demonstrations to self-study assignments.

The EPOSS architecture foresees three domains of authority: student, lecturer/educator, and network/cloud. In on-site courses, these authorities can be combined, whereas in remote teaching they are typically disjoint. In the cloud domain, administered by the educator or the IT department of the responsible institution, network and proxy services are operated to facilitate the connection between students and educators. Students perform requests containing programming instructions to the proxy service which are then executed in the educator domain that also encompasses the physical 'things'. Fig. 3 compares the teaching settings.

The architecture takes security and privacy concerns into account by forwarding all service requests
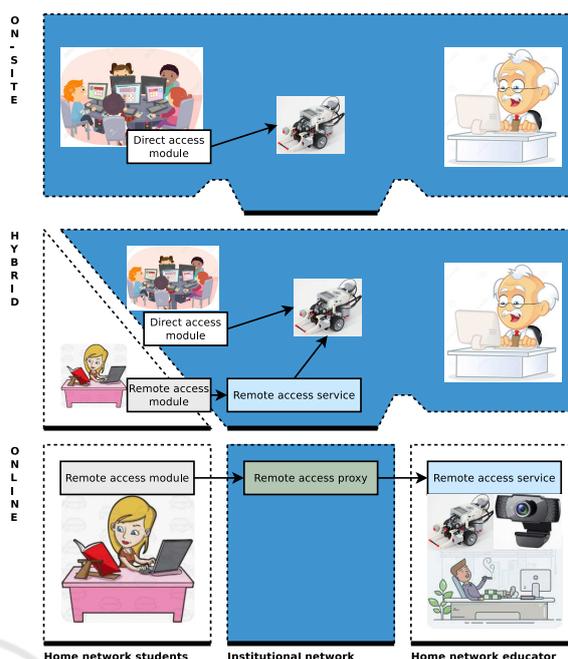

Figure 3: EPOSS teaching settings.

through a message queue. Hence, the educator does not need to expose any network ports directly to the outside world in hybrid and online teaching settings. Furthermore, devices can be configured to register at the proxy upon booting with a randomly generated secret number that is spoken via voice synthesis for two-factor authentication. This feature can be used not only to authorise access across sites, but also to avoid accidental or malicious takeover across classrooms. Only students who are nearby and listen to the voice are thus able to control the device. Moreover, as all interactions traverse the proxy, all programming instructions can be recorded and later be inspected or replayed.

4.2 Implementation

The implementation makes assumptions about the programming language and the interfaces to 'things' as well as the operating environment and the covered programming domains. While the implementation technologies are generally exchangeable, the following describes the implemented EPOSS architecture that assumes the use of Python with RPC interfaces and the network protocols HTTP and AMQP. The proxy-connected Python module RPyCloud is at the centre of the solution along with the corresponding RPyCloud service. It is a drop-in replacement for the standard module RPyC used for local device programming, making it possible to re-use programming

instructions without modification. The RPyCloud device code is optional; for 'things' that do not permit any modification, it does not have to be used at the expense of not being able to isolate the devices between students.
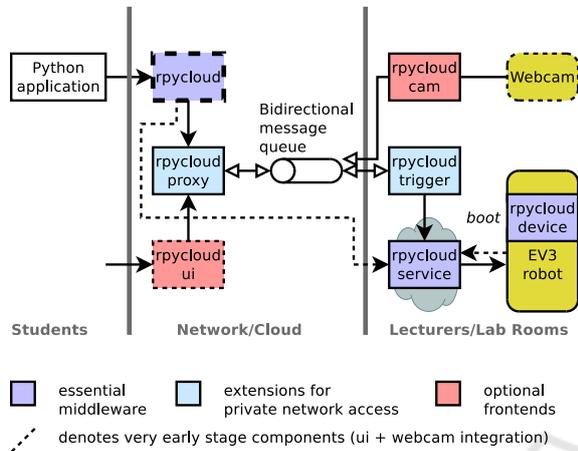

Figure 4: RPyCloud architecture and implementation.

Fig. 4 shows the implementation-level architecture with components spread across the three domains of authority. The invocations that are channeled to the service are executed on the device. Hence, for devices requiring other forms of interactions, such as HTTP or ROS messages, the module and the execution part within the service need to be adjusted to maintains the seamless substitution of standard interfaces. This way, the education setup does not deviate significantly from the interfaces that students will use in their careers after graduation.

An EPOSS should furthermore contain hands-on lab examples that can be used in supervised/guided or unsupervised/self-study settings. The implementation therefore contains specific examples for vehicle programming for traffic systems that are described below.

4.3 Interaction

Using the support structure for direct access to enforce exclusive interaction with the programmable things does not involve any special effort other than using the RPyCloud module by students and operating the corresponding server-side infrastructure. In contrast, remote access requires a proper setup with cameras, microphones and other sensors so that the senses of students get activated. Fig. 5 shows a typical physical setup using an external camera so that students can interact with both the things and the educator side-by-side.
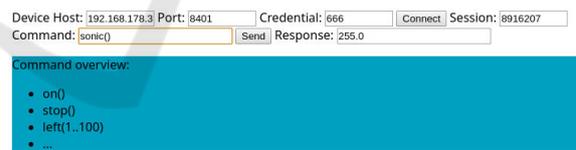

Figure 5: Physical setup in classroom or educator home.

As feedback about the consequences of a programming instruction is crucial, a web-based command and control interface is furthermore operated and made accessible through the cloud domain. It allows for connecting to a specific device and submission of individual abstractions, but can also be used in parallel to a running program to visibly and audibly verify the correctness and behaviour. While it enables self-study, it can also be used interactively to perform a discussion between student and educator. This reduces the need for a separate chat and video call tool and thus reduces the cognitive load on students.

Fig. 6 shows a typical remote programming session in which a lab instructor explains to a student how to correct a mistake.

Figure 6: RPyCloud Web user interface.

5 USE CASE: CYBER-PHYSICAL VEHICLE PROGRAMMING

Intelligent transport systems are a cornerstone of smart cities and regions. Students of traffic systems and traffic engineering learn about transport modalities and associated concerns like logistics and integrated mobility. The EPOSS is suitable for teaching the programming of vehicle movements and transport schedules in on-site, hybrid and remote settings. At the author's institution, these students learn imperative, procedural and object-oriented programming concepts over a period of two semesters in cohorts of up to 40 students. The competences are conveyed with the Python programming language and its diverse modules for interacting with the computer as well as external systems. Hands-on programming labs are conducted individually or in pairs and are geared towards the students' future work domains, for instance intelligent and data-supported traffic systems development. While the first two weeks are reserved for foundational theory (introducing programming, language syntax), the programming of robotic vehicles starts as early as in the third week, and in the second semester extends to smart city scenarios including avoidance of accidents in intermodal traffic situations. The use of physical 'things' aids in a more plastic and realistic representation of these scenarios.

5.1 Vehicle Scenario Installation

Fig. 7 shows a physical model of a crossing of two roads, deployed in the classroom in on-site or hybrid settings, or in the homes of students or educators in pure online teaching settings. Two model vehicles with different characteristics, primarily speed and offensive-defensive driving styles, along with three measurement points and a number of additional sensors and light indicators are connected to form a crossing scenario. All vehicles and measurement points are represented by Mindstorms EV3 robots that fit the RPC programming model assumed by EPOSS.

5.2 Manoeuvre Programming

To foster the mastering of algorithms involving precise temporal and spatial aspects, students can use the EPOSS to implement traffic manoeuvres typical for cars on roads. Educators have access to reference implementations which the students can also replay on their own account to compare their own solutions.

Fig. 8 shows the implemented scenarios involving either one or two vehicles.
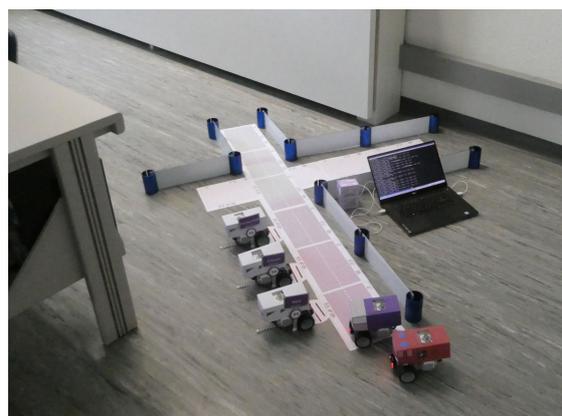

Figure 7: Traffic counting scenario with moving vehicles and stationary sensors.
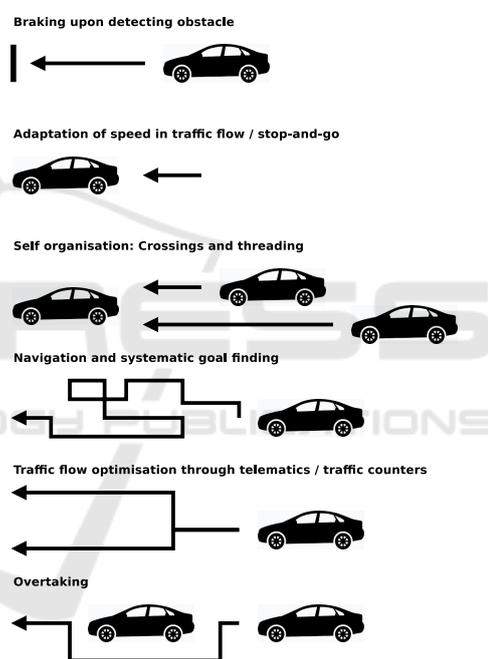

Figure 8: Vehicle manoeuvres.

The programming is non-trivial as can be exemplified with the overtaking manoeuvre. Overtaking requires a stateful multi-step process. In the first step, upon detecting a slow vehicle in front, the decision to overtake is made by turning to the left or, in case the vehicle robots do not permit rotations as is the case with EV3 robots, increasing the acceleration of the right wheel. In the second step, the sides are reversed to achieve a right turn. In the third step, both sides are accelerated equally. By programming actual 'things', limited sensor information must be considered and mistakes may lead to actual crashes that stress the importance of safe algorithms in spite of their complexity.

5.3 Open Data and Autoprogramming

In order to achieve realistic scenarios, real-time traffic data can be injected into the physical model. For this use case, the EPOSS has been extended with access to two systems, OTD and VDP from Switzerland. OTD delivers planning and actual data with minute granularity on public transport, whereas VDP delivers actual data with second and even subsecond granularity on road vehicles. Both systems intersect when public transport is using the road, in particular buses.

Fig. 9 shows the general concept of injecting and replaying open data in the physical model. It is not only increasing the attractiveness for students due to the stronger link between model and real system. Rather, it also allows for reasoning about the precision of all system components ranging from real system sensing over physical movement and sensing in the physical model to digital data processing of all sensor data and programming instructions. This exposes students to the open research problem of achieving absolute synchrony, represented by the same deltas between measurements in both systems, and can lead to follow-up engagement in project works on tackling parts of this problem.
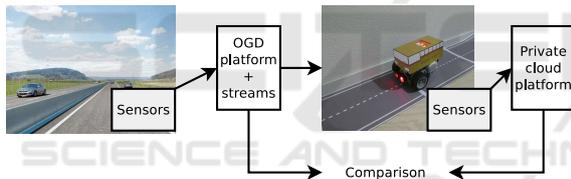

Figure 9: Replaying data captured from real systems in model systems.

Fig. 10 shows how the movement of a scheduled train visible in the background is projected in seemingly perfect synchrony to the movement of a model train. This aspect is particularly intriguing and attractive for students who understand the complexities of traffic planning. In this case, as OTD only delivers minute granularity, the students have to perform statistical measurements to calculate movement drifts and reason about increasing the precision and predictability in mobility.

5.4 Student Feedback

EPOSS is work in progress and was only used in a pilot setting in the first of two semesters with traffic engineers. It was not yet evaluated in other engineering domains. Hence, only basic vehicle manoeuvres such as drawing a circle or a square with appropriate movement sequences were taught with the support structure. The flexibility in settings proved
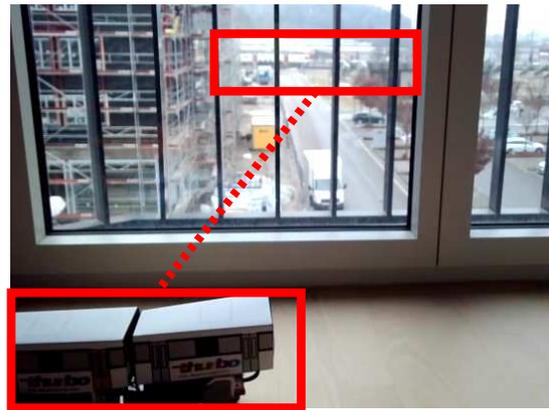

Figure 10: Autoprogramming of a model train with OTD from a real train.

to be an advantage, as the teaching started in hybrid mode (with on-site and remote groups taking turns) and later switched to pure online mode. Accordingly, the physical installation was migrated from the classroom to the educator home office, without any work difference for those students who were previously in the remote groups. The general perception of the system was positive and the interaction worked well, although the participants look forward to enhancements such as self-reservation for unsupervised programming. Compared to a classroom, the supervised 1:1 sessions were considered especially helpful in case of problems like stuck cars that could quickly be resolved by the intervention of the educator.

6 CONCLUSIONS

EPOSS is a digital support system for educational needs related to programming and observing physical objects or 'things'. It is implemented based on a distributed computing architecture and proven to work in on-site, hybrid and remote teaching of traffic engineering students. The focus of the work was on the technical enablement. Therefore, it lacks empirical validation over multiple semesters. This task is left for future work from a scientific work angle. To extend EPOSS to other study courses, the physical scenery could be extended to support multiple cameras and ground materials (useful for machine technology students) as well as cranes and other logistics equipment (useful for business process students). Specifically for traffic engineering students, intermodal logistics scenarios including handover of goods and enhancement with simulation to learn the behaviour of complex systems and scale could be realised. These extensions are left as further future work from an applied education angle.

EPOSS AVAILABILITY

EPOSS is made available as open source software package, consisting of the central RPyCloud system, traffic demonstration scenarios and other necessary parts to replicate it in other education institutions. A snapshot is published on Zenodo[1] while further development is encouraged on GitHub[2].

ACKNOWLEDGEMENTS

REFERENCES

Almeida, T. O., de Magalhães Netto, J. F., and Rios, M. L. (2017). Remote robotics laboratory as support to teaching programming. In *2017 IEEE Frontiers in Education Conference, FIE 2017, Indianapolis, IN, USA, October 18-21, 2017*, pages 1–6. IEEE Computer Society.

Altuger-Genc, G., Han, Y., and Genc, Y. (2018). Towards simulation aided online teaching: Material design for applied fluid mechanics. *Int. J. Online Eng.*, 14(12):112–125.

Bolano, G., Rönnau, A., Dillmann, R., and Groz, A. (2020). Virtual reality for offline programming of robotic applications with online teaching methods. In *17th International Conference on Ubiquitous Robots, UR 2020, Kyoto, Japan, June 22-26, 2020*, pages 625–630. IEEE.

dos Santos, M. T., Vianna Jr, A. S., and Le Roux, G. A. (2018). Programming skills in the industry 4.0: are chemical engineering students able to face new problems? *Education for Chemical Engineers*, 22:69–76.

Guo, P. (2013). Why Scientists and Engineers Must Learn Programming. Communications of the ACM Blog.

Hodges, S., Sentance, S., Finney, J., and Ball, T. (2020). Physical computing: A key element of modern computer science education. *Computer*, 53(4):20–30.

Karaimer, H. C. and Brown, M. S. (2018). Improving color reproduction accuracy on cameras. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 6440–6449. IEEE Computer Society.

Klimova, B. F. and Poulova, P. (2014). Pedagogical issues of online teaching: Students' satisfaction with online study materials and their preferences for a certain type. In Cao, Y., Väljataga, T., Tang, J. K. T., Leung, H., and Laanpere, M., editors, *New Horizons in Web Based Learning - ICWL 2014 International Workshops, SPeL, PRASAE, IWMPL, OBIE, and KMEL, FET, Tallinn, Estonia, August 14-17, 2014, Revised Selected Papers*, volume 8699 of *Lecture Notes in Computer Science*, pages 187–194. Springer.

Meyer, R. A. and Seawright, L. H. (1970). A virtual machine time-sharing system. *IBM Syst. J.*, 9(3):199–218.

Nortvig, A., Petersen, A. K., Helsinghof, H., and Brænder, B. (2020). Digital expansions of physical learning spaces in practice-based subjects - blended learning in art and craft & design in teacher education. *Comput. Educ.*, 159:104020.

Ocaña, J. M., Urrutia, E. K. M., Pérez-Marín, D., and Pizarro, C. (2020). Can a learning companion be used to continue teaching programming to children even during the COVID-19 pandemic? *IEEE Access*, 8:157840–157861.

Ortiz, O., Alcover, P. M., Sánchez, F., Pastor, J. A., and Herrero, R. (2015). M-learning tools: The development of programming skills in engineering degrees. *Rev. Iberoam. de Tecnol. del Aprendiz.*, 10(3):86–91.

Peter, S., Momtaz, F., and Givargis, T. (2015). From the browser to the remote physical lab: Programming cyber-physical systems. In *2015 IEEE Frontiers in Education Conference, FIE 2015, El Paso, TX, USA, October 21-24, 2015*, pages 1–7. IEEE Computer Society.

Prinsley, R. T. and Baranyai, K. (2013). *STEM skills in the workforce: what do employers want?* Office of the Chief Scientist.

Santana, B. L., Figuerêdo, J. S. L., and Bittencourt, R. A. (2018). Motivation of engineering students with a mixed-contexts approach to introductory programming. In *IEEE Frontiers in Education Conference, FIE 2018, San Jose, CA, USA, October 3-6, 2018*, pages 1–9. IEEE.

Smith, A. C. (2007). Using magnets in physical blocks that behave as programming objects. In Ullmer, B. and Schmidt, A., editors, *Proceedings of the 1st International Conference on Tangible and Embedded Interaction 2007, Baton Rouge, Louisiana, USA, February 15-17, 2007*, pages 147–150. ACM.

Thode, E., Schleiter, A., and da Silva Zech, G. (2020). Digitalization permeates the entire working world – Eight in 10 German Occupations Now Demand Digital Skills. Study by Burning Glass Technologies and Bertelsmann Stiftung.

Vázquez-González, J. L., Barrios-Aviles, J., Muñoz, A. R., and Palomares, R. A. (2018). An industrial automation course: Common infrastructure for physical, virtual and remote laboratories for PLC programming. *Int. J. Online Eng.*, 14(8):4–19.

---

[1]EPOSS snapshot: https://doi.org/10.5281/zenodo.4547808

[2]EPOSS development: https://github.com/serviceprototypinglab/eposs