A Multi-Interface Platform for Teaching Sensor Systems Using Unity, Web, and ROS Interfaces

A. Baraka*[a], L. Manassés Pinheiro de Souza[a], A. Göppert[a], and R. Schmitt[a,c]

[a]Chair for Intelligence in Quality Sensing (IQS), WZL, RWTH Aachen University, Germany
[b]Fraunhofer Institute for Production Technology IPT

ABSTRACT

Practical experience with sensor systems is essential in engineering education, yet providing scalable laboratory environments for large student groups remains challenging. This work presents a multi-interface platform designed to teach sensor systems through progressive interaction with virtual and physical systems. The platform integrates sensors, microcontrollers, network communication, and interactive interfaces. Students connect sensors such as Inertial Measurement Units (IMUs), rotary encoders, and infrared sensors to microcontrollers. The measured signals are transmitted via wireless communication to external interfaces where they control system behavior. The platform supports three complementary interfaces. First, the Unity Interface allows intuitive visualization and rapid experimentation with sensor-driven interaction in a safe virtual scene. Second, the Web Interface enables distributed interaction with a shared digital factory scene accessible through standard web browsers. Finally, the Robot Operating System Version 2 (ROS) interface extends the same sensing concepts to robotic simulation and real robotic hardware, where sensor signals influence the motion and behavior of robotic arms. The proposed platform enables a progressive learning approach in which students move from virtual experimentation to real-world robotic interaction. It demonstrates how modern sensor education can combine simulation and physical robotics within one unified platform.

Keywords: Sensor Systems, Measurement Systems Education, Unity Interface, Web Interface, ROS Interface, Robotic Arms

1. INTRODUCTION

Sensor systems are central to modern engineering applications, where measured physical quantities are used to control digital processes, connected machines, and robotic systems. In many practical applications, a single sensor is not sufficient to describe the state of a system or the surrounding environment. Instead, information is often acquired from multiple sensors or data sources and then combined to obtain a more reliable, accurate, and meaningful understanding of the process being observed. Examples include robotic systems that combine inertial, position, proximity, force, and visual information to estimate motion, detect objects, and react to changing conditions. Therefore, engineering students should understand not only the physical principles of individual sensors, but also the complete information chain from signal acquisition and data transmission to sensor interpretation, data fusion, and system response. This makes sensor systems education inherently practical. Students need to connect sensors, acquire measurement signals, process data on hardware, transmit information through communication interfaces, and observe how measured data influence the behavior of a technical system.

Traditional laboratory exercises provide valuable hands-on experience, but they are often difficult to scale for large student groups. Physical laboratory setups require space, supervision, maintenance, safety measures, and repeated access to hardware. These constraints become more significant when courses include many students working in parallel or when the learning objectives involve robotic systems, network communication, and industrial automation equipment. Therefore, sensor systems courses require teaching concepts that preserve the benefits of practical work while improving accessibility, flexibility, and scalability. The broad range of skills addressed by the proposed project is summarized in Figure 1. The project supports a progressive learning path from foundational skills and engineering practice to sensor system implementation, simulation tools, and advanced robotic applications.

*Corresponding author: A. Baraka, ahmed.baraka@wzl-iqs.rwth-aachen.de

IMEKO PHOTOMET EDUMET 2026 Metrology in Transition: Education, Photonics, and the Impact of AI and Robotics, edited by Hubert Zangl, Maik Rosenberger, Luigi Rovati, Proc. of SPIE Vol. 14495, 144950X · © The Authors.
Published under a Creative Commons Attribution CC-BY 4.0 License · doi: 10.1117/12.3132689

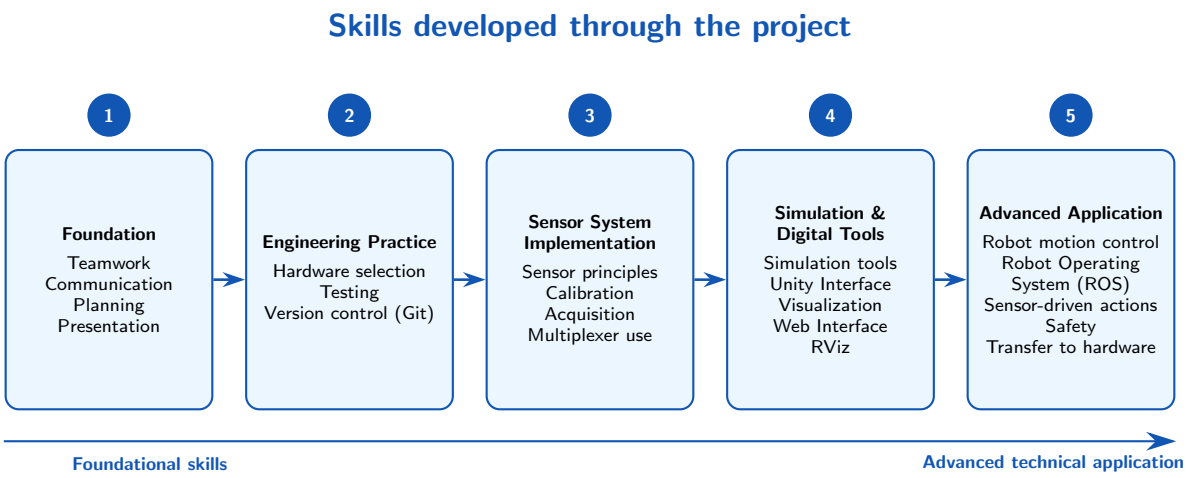


Figure 1. Progressive development of engineering and technical skills through the proposed platform.

The need for practical sensor experience, scalable course organization, and accessible interaction with robotic and automation scenarios motivates a teaching platform that combines real sensor hardware with scalable digital interfaces instead of treating hardware experiments, simulation exercises, and robotic applications as separate course elements. The central idea of this work is to preserve the practical value of wiring, programming, and hands-on sensor testing while making the resulting interaction accessible through interfaces that can be used by many student groups in parallel. In this structure, students first build a physical sensor system and then use the measured values to influence external systems of increasing complexity.

Building on this motivation, the platform is organized around a common sensor interaction chain that connects physical sensor acquisition, local processing, data transmission, application-level interpretation, and observable system response. The same chain can be used with different sensors, microcontroller boards, and communication setups, which allows the practical depth of the assignment to be adapted to student experience, available time, and laboratory resources.

The contribution of this work is the integration of three complementary interfaces into one continuous learning concept. First, the Unity Interface supports safe and intuitive experimentation with sensor-driven interaction in a visual scene. Second, the Web Interface extends the concept toward distributed interaction with a shared digital factory scene through standard browsers. Third, the Robot Operating System Version 2 (ROS) Interface, hereafter referred to as the ROS Interface, connects the same sensing and communication concept to robotic simulations and selected physical robotic systems, where student-generated sensor signals influence robotic motion and actions. These interfaces are used as progressive stages of the same teaching platform rather than as independent exercises.

The platform was applied in two educational settings. The Unity Interface and the ROS Interface were implemented directly in the master's course "Robotic Sensor Systems," offered by the Chair for Intelligence in Quality Sensing (IQS) at the Laboratory for Machine Tools and Production Engineering (WZL), RWTH Aachen University. Across two semesters, these course implementations involved more than 200 students working in approximately 50 groups. The Web Interface was applied separately in a summer school with students at a partner university.

The remainder of this paper is organized as follows. Section 2 discusses related work and identifies the gap addressed by the proposed platform. Section 3 presents the overall system architecture. Section 4 describes the Unity Interface, the Web Interface, and the ROS Interface. Section 5 discusses student implementations, Section 6 presents the educational evaluation, and Section 7 concludes the paper and outlines future work.

2. RELATED WORK

Practical engineering education is strongly connected to active learning, collaborative work, and project-based learning. Prince showed that active, collaborative, cooperative, and problem-based learning approaches are

supported by broad evidence in engineering education, although their effectiveness depends on suitable implementation and course design.[1] In sensor systems education, this is particularly relevant because students must not only understand physical sensor principles, but also work on technical problems involving hardware integration, communication, signal processing, and observable technical reactions.

Virtual and remote laboratories have been proposed as one solution to the scalability and accessibility limitations of conventional laboratory teaching. Heradio et al. showed through an analysis that virtual and remote laboratories have become an established research field in education, with strong relevance for engineering and science fields.[2] More recent studies confirm this trend. Li and Liang conducted an analysis on virtual laboratories in engineering education and reported positive effects on student learning performance.[3] Wahyudi et al. also emphasized that virtual laboratories can provide safe, economical, and flexible learning environments, while remaining most effective when their limitations are considered and when they complement practical work rather than fully replace it.[4]

Remote laboratories are especially relevant when students need access to hardware that is limited, expensive, or difficult to supervise continuously. Van den Beemt et al. studied remote laboratories in higher engineering education and connected them to active learning.[5] In the context of embedded systems and IoT education, Jacko et al. presented a remote IoT education laboratory for microcontroller teaching, combining physical microcontroller work with remote supervision and monitoring.[6] More recent IoT remote laboratory concepts, such as VREL NextGen, continue this direction by enabling students to interact with microcontrollers, sensors, actuators, and visual feedback.[7] These works demonstrate the value of connected hardware access, but they often focus on predefined laboratory setups rather than on student-developed sensor systems that can be transferred across several interaction interfaces.

Digital twins and virtual laboratories have also become important in industrial and robotics education. Terkaj et al. presented virtual laboratories for higher education in industrial engineering and showed how digital representations of manufacturing laboratories can support accessible learning.[8] In robotics education, Singh et al. demonstrated the use of Unity and ROS as digital and communication components for robotic arm digital twin applications in smart manufacturing cells.[9] Similarly, Pantusin et al. highlighted the value of digital twin integration for active learning in robotic manipulator control.[10]

Game engines and real-time visualization platforms are increasingly used in this area because they provide intuitive interaction with simulated technical systems. Interfaces developed with Unity can visualize robot motion, sensor effects, and process behavior in a way that helps students connect abstract data with observable system reactions. This makes such interfaces useful not only for programming education, but also for teaching automation, control, and robotic system behavior. However, many existing approaches focus on a specific robot, machine, or virtual laboratory. Less attention is given to flexible teaching platforms where students can connect different sensor types and microcontroller configurations to several interfaces using the same communication concept.

Collaboration is another important dimension of engineering laboratories. Kurtz et al. reviewed collaboration in virtual and remote laboratories and found that communication and group awareness tools support collaborative learning by helping students share knowledge and understand the actions of other group members.[11] This is directly relevant for large sensor systems projects, where students must coordinate hardware integration, software development, testing, troubleshooting, and demonstrations. Browser-accessible platforms can further support this by allowing several student groups to interact with a shared environment without requiring identical local hardware setups.

Overall, the literature shows strong progress in virtual laboratories, digital twins, and robotics education. These approaches improve accessibility, safety, scalability, and student engagement. Nevertheless, many existing systems address these aspects separately. Virtual laboratories often emphasize simulation, remote laboratories give access to predefined hardware, and robotic digital twins frequently focus on a specific platform or experiment. In addition, many educational platforms are designed around fixed laboratory structures, predefined hardware configurations, or narrowly defined learning objectives. The gap addressed in this paper is therefore not the use of one additional virtual or remote laboratory, but the integration of student-built sensor systems with several interaction interfaces through one reusable sensing and communication concept.

3. SYSTEM ARCHITECTURE

The proposed teaching platform is built around a common sensor interaction chain that connects physical sensors with digital and physical system responses. The basic workflow is organized into five clearly separated stages: data acquisition, local processing, data transmission, application-level interpretation, and system response. This structure allows students to experience the complete path from a measured physical quantity to an observable action in an external environment. At the same time, the architecture remains flexible enough to support different sensor types, microcontroller boards, communication methods, and levels of implementation complexity.

The architecture is designed as a modular platform rather than a fixed laboratory setup. Students can work with simple sensor configurations in introductory implementations or combine several sensors and communication functions in more advanced projects. The same basic structure can be used to control a virtual simulation, interact with the Web Interface, or extend the project to physical robotic hardware. This modularity is central to the course concept because it allows the practical work to be adapted to the available time, number of students, laboratory resources, and course objectives.

Figure 2 illustrates the overall architecture of the platform. The architecture is read from left to right: physical quantities are first captured by sensors, the resulting signals are acquired by an embedded device, the values are processed into application-relevant control variables, and these variables are transmitted to one of the interaction interfaces. Depending on the implementation stage, the received data can influence the Unity Interface, the Web Interface, or the ROS Interface with optional physical robotic hardware. In all cases, students can directly observe the effect of their sensor system on the behavior of a technical system.

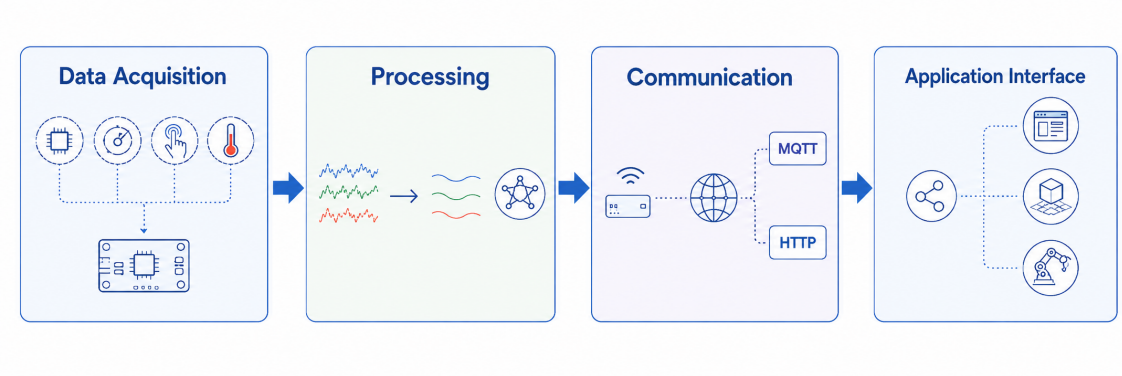


Figure 2. Overall architecture of the multi-interface sensor systems platform.

3.1 Data acquisition module

The first module of the platform is the data acquisition module. It includes the physical sensors and immediate acquisition electronics used by student groups to capture physical quantities. Its role is limited to producing reliable raw or low-level sensor values; the interpretation of these values and their transfer to external interfaces are handled by the following modules. This separation makes the architecture easier to understand because students can first focus on how physical phenomena become measurable data.

The sensor selection can be adjusted depending on the intended learning outcome and the required project complexity. Typical sensors include inertial measurement units (IMUs), rotary encoders, infrared sensors, capacitive or touch sensors, piezoelectric elements, and temperature sensors. These components allow students to work with different physical quantities such as acceleration, orientation, rotation, proximity, contact, force related signals, and temperature.

The sensors can produce different types of input signals. Some sensors provide simple digital states, for example whether an object is detected or whether a touch event occurred. Other sensors provide continuous analog or digital measurement values, such as acceleration, angular velocity, rotation count, or temperature. This variation is useful from an educational perspective because students learn that different sensor principles also require different acquisition and interpretation strategies.

The sensor modules are connected to embedded computing platforms such as ESP8266, ESP32, Arduino boards, or Raspberry Pi based systems. In this module, these platforms primarily serve as acquisition devices: they provide analog and digital inputs, sensor buses, timing functions, and a practical connection point between the sensors and the rest of the system. The choice of platform can be adapted to the intended level of complexity. Simple microcontroller boards are sufficient for basic signal acquisition, while more advanced platforms can be used when additional inputs, image acquisition, or higher sampling requirements are needed. This allows student groups to build independent prototypes that can be tested, demonstrated, and modified without requiring permanent access to large laboratory equipment.

In addition to the core sensors, the data acquisition module can be extended with further components depending on the course scope. For example, students may include additional sensors, mechanical mounting elements, custom housings, or camera systems. Displays and actuators can also be added when they support local testing, but they are not required for the basic acquisition chain. The integration of a camera can extend the project toward basic machine vision concepts, such as image acquisition, object detection, feature extraction, or visual feedback. This makes the acquisition setup suitable for both basic sensor exercises and more open-ended project implementations.

3.2 Processing module

The processing module converts the acquired sensor data into meaningful information that can be used by the platform interfaces. This module is separate from acquisition because it does not describe which sensor is connected, but how the measured values are cleaned, calibrated, combined, or transformed into control variables. This step is important because the platform is not limited to reading sensor values. Instead, students must decide how measured signals should be interpreted and how they should affect the behavior of the system.

For simple sensors, the processing may consist of detecting state changes or threshold events. An infrared sensor, for example, can be interpreted as a trigger when an object is detected. A capacitive touch sensor can be used as a binary input for activating a function. For continuous sensors, the processing can include scaling, filtering, calibration, or conversion into physical or logical quantities. Rotary encoder pulses can be converted into angular displacement or relative position. Inertial sensor values can be used to estimate orientation or movement direction. Temperature readings can be interpreted as process conditions or used to activate specific system responses.

Depending on the student implementation, the processing can be performed on the microcontroller, on a single-board computer, or in a connected software component. The processing unit may also combine several sensor inputs into one interaction logic. For example, one sensor may determine movement direction while another sensor triggers an action. This supports the educational goal of moving from isolated sensor measurements toward complete sensor systems.

The interpretation of sensor data can also be extended through simple machine learning approaches that classify sensor patterns, recognize gestures, detect system states, or combine several sensor signals into higher-level decisions. For example, inertial sensor data could be used to recognize basic motion patterns, while camera-based input could introduce simple neural network concepts for visual classification or object detection. These extensions are not required for the basic platform implementation, but they show how the same architecture can support more advanced teaching topics such as machine learning, sensor fusion, and machine vision.

The processing module also introduces students to practical engineering challenges that are often not visible in purely simulated exercises, such as unstable signals, noisy measurements, timing jitter, sensor calibration, and wiring errors. By solving these issues in their own prototypes, students gain experience with the practical constraints of real sensor integration before the data are transmitted to an external interface.

3.3 Communication module

After the sensor data have been acquired and processed, the resulting control variables are transmitted to an external interaction interface. The communication module is therefore responsible only for representing and transporting data between the student prototype and the selected interface. This separation keeps the hardware and processing logic independent from the application interface and allows the same sensor system to be reused across different interaction stages.

Depending on the project setup, the communication can be implemented through wired or wireless methods. In early implementations, a USB serial connection between the microcontroller and a computer can be sufficient to transmit sensor values to a local application. In more advanced implementations, the microcontroller can transmit data wirelessly to the Web Interface, a local network service, or another software component that receives and interprets the incoming messages. Different communication protocols can be introduced depending on the required complexity. Simple serial messages may be sufficient for local demonstrations, while HTTP requests, WebSocket communication, TCP or UDP messages, or MQTT can be used for networked implementations.

The transmitted data can be formatted as simple command values, numerical sensor readings, or structured messages containing several sensor states. The exact format can be kept simple for introductory groups or extended for advanced groups. This allows students to understand communication as a functional part of the system architecture rather than as an isolated programming task. They learn that sensor data must be represented in a clear way before they can be used by another software or hardware component.

The communication module also supports scalability and creative variation within the course. Different student groups can develop different hardware solutions that still interact with the same platform interface. This allows groups to explore different sensor combinations, interaction ideas, and control strategies while remaining within a common technical structure. As a result, the platform does not force all groups to follow the same implementation path, but it encourages open-ended problem solving and creative engineering solutions.

3.4 Application interface module

The application interface module receives the transmitted sensor information and maps it to actions inside an external interface. It is responsible for interpreting incoming data in the context of the selected interface and converting it into movement, interaction, process control, or robotic action.

In the Unity Interface, the application interface module can map sensor data to the behavior of virtual objects or characters. For example, orientation or movement signals can be used to control motion, while trigger signals can activate interactions in the simulated scene. In the Web Interface, the same type of sensor information can be used to control digital factory processes, shared visual elements, or group interactions. In the ROS Interface, sensor information can be transferred to simulated or physical robotic actions when the required safety and laboratory conditions are available.

The important architectural point is that the interfaces are not independent teaching exercises. They are different application targets for the same sensing and communication concept. This allows students to first develop and validate their sensor logic in accessible digital interfaces before moving to more complex or resource intensive implementations. The platform therefore creates a gradual transition from virtual experimentation to distributed interaction and, where possible, physical robotic response.

3.5 Supporting software tools and development workflow

The previous subsections describe the functional modules of the architecture. In addition, the platform introduces students to a broader software and engineering workflow that supports these modules. The required tools depend on the selected implementation depth, but the architecture can support several complementary skill areas. Microcontroller programming is typically carried out using the Arduino IDE or comparable embedded development environments. In this stage, students work with C or C++ style programming to acquire sensor signals, process values, define thresholds, and prepare data for communication.

For the Unity Interface, Unity can be used together with C# scripts to define the behavior of virtual objects, interaction logic, and real-time visual feedback. This allows students to connect sensor data with a visual and interactive system. For the Web Interface, students can also be exposed to web technologies such as HTML, CSS, and JavaScript, depending on how much of the platform implementation is included in the student tasks. These tools are especially relevant when sensor values are transmitted to interfaces that can be accessed by several users or groups.

Python can be added as an optional tool for data analysis, prototyping, visualization, or additional processing tasks. This can be useful when students need to inspect sensor data, evaluate signal behavior, or develop small

helper scripts. Version control with Git can also be integrated into the project workflow to support group collaboration, documentation, and structured software development.

For robotic interaction and advanced automation scenarios, the platform can also be extended toward robotics open-source software middleware such as ROS. In such implementations, sensor information can be forwarded to ROS nodes, where it can be integrated into robotic control pipelines, visualization environments, or simulated robotic systems. Tools such as RViz can be used to visualize robot states, sensor information, coordinate systems, and motion behavior in real time, while robotic simulation environments support testing before interaction with physical hardware.

Containerization tools such as Docker can further support reproducible software environments and simplify deployment across different student systems or laboratory computers. This is especially useful in larger robotics projects where several software dependencies, communication services, or simulation components must operate consistently across multiple development environments.

The platform can further be extended beyond software development. Students may design mechanical holders, housings, or integration parts using computer aided design tools. In more advanced versions of the project, printed circuit board (PCB) design tools such as KiCad can be introduced to move from breadboard prototypes toward more integrated electronic systems. These extensions are not mandatory for every course implementation, but they show how the same platform can support a wider engineering workflow from sensor prototype to integrated system design.

This flexibility is important because it allows instructors to adapt the project to different educational goals. A small implementation can focus mainly on sensor wiring and microcontroller programming. A more advanced implementation can include simulation, wireless communication, collaborative software development, mechanical design, or circuit board design. As a result, the platform can be scaled from a basic sensing exercise to a broader interdisciplinary engineering project.

3.6 Scalability and adaptability

The architecture was designed to support different levels of complexity without changing the fundamental learning concept. At the simplest level, a student group can connect one sensor to a microcontroller, read the signal, and use it to trigger a visible response in a digital environment. At a more advanced level, several sensors can be combined, processed locally, transmitted wirelessly, and used to control complex interactions in a shared virtual or physical system.

This scalability is useful for large courses because the technical depth of the project can be adjusted while the overall platform structure remains unchanged. Depending on the course objectives, available time, laboratory resources, and intended learning outcomes, the project can focus on basic sensor acquisition and simple control logic, or it can be extended toward additional communication functions, multiple sensor inputs, more complex interaction logic, and physical prototype integration.

The modular structure also reduces the dependence on expensive or limited equipment. The Unity Interface and the Web Interface can be used by many groups in parallel, while the physical robotic system can remain an optional extension of the ROS Interface for selected demonstrations or advanced project stages. This makes the concept suitable for courses with different numbers of students and different levels of laboratory access. The robotic setup becomes the physical continuation of the same interaction chain rather than a separate mandatory resource for all groups.

Another advantage of the architecture is that it can evolve over time. New sensors, microcontrollers, software tools, or interaction interfaces can be added without changing the basic concept. This allows the platform to be reused and further developed in future course iterations. It also gives instructors the possibility to adapt the project to new topics such as sensor fusion, IoT applications, digital twins, collaborative automation, or robotic control.

3.7 Summary

The system architecture provides a common technical foundation for the proposed multi-interface teaching platform. It connects physical sensor acquisition, processing, communication, interpretation, and system response within one continuous workflow. This enables students to experience the complete information chain of a sensor system while allowing the implementation complexity to be adapted to the course context.

The next section describes how this architecture was implemented in the three interaction interfaces: the Unity Interface, the Web Interface, and the ROS Interface.

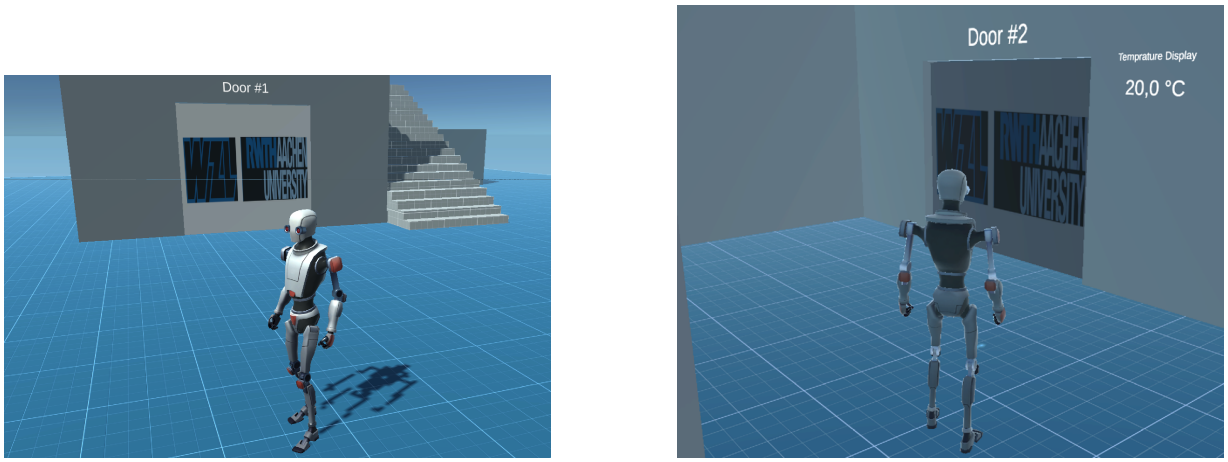
4. IMPLEMENTATION INTERFACES

The proposed platform was implemented through three complementary interfaces that represent different levels of technical complexity. Each interface uses the same general principle introduced in the system architecture: sensor data are acquired from student developed hardware, processed, transmitted to an interface, and converted into an observable system response. The interfaces were developed using accessible software tools that do not require additional license fees. This allows students to use the required tools on their own devices, while also supporting reuse of the platform in future course iterations and in other teaching contexts.

The three interfaces are not intended as independent exercises. Instead, they form a progressive implementation path. The Unity Interface provides an accessible introduction to sensor-driven interaction in a visual scene. The Web Interface extends this concept toward distributed interaction with an industrial process scenario. The ROS Interface then transfers the same sensing and communication concept to robotic simulation and physical robotic systems.

4.1 Unity Interface

The first implementation interface was developed in Unity as an accessible simulation stage for sensor-driven interaction. The interface consisted of a predeveloped open world game scene in which students could control a humanoid robot character using physical sensor inputs. The scene included a house with two rooms and three doors, creating a simple but flexible interaction space for movement, navigation, door control, and environmental feedback. Figure 3 shows two views of the Unity Interface developed for the course.



(a) (b)

Figure 3. Unity Interface used for sensor-driven control of a humanoid robot character: (a) open-world scene for navigation and interaction tasks; (b) interior scene used for room-based interaction and visual feedback.

The Unity Interface was designed to give students an immediate visual response to their sensor systems. Instead of observing sensor values only through a serial monitor, students could see how their measured signals affected the behavior of a digital system. The required interactions included regular robot movement, running, jumping, camera movement, door control, and the display of room temperature inside the game scene. Students

were also required to include filtering of sensor signals, which connected the visual interaction with practical signal processing challenges such as noise reduction, unstable readings, and delayed response.

The implementation followed planned progressive milestones. In the first milestone, students installed the required software, ran the Unity project, and examined the existing game code. In parallel, they tested the microcontroller using simple Arduino code. Next, the sensors were connected to the microcontroller and their values were displayed in the Arduino serial monitor or serial plotter. Wireless communication was then introduced by transmitting a simple test message from the microcontroller to Unity. After this basic communication test, the next milestone was to replace this message with real sensor data that had to be formatted correctly and parsed inside Unity. Finally, the sensor readings were mapped to the game mechanics, including robot movement, door actions, real-time temperature display, and in-game feedback.

Unity was also selected because real-time 3D visualization tools are increasingly relevant beyond teaching. Such tools are used in engineering research and development for digital twins, simulation, interactive visualization, prototyping, and robotics related applications. Recent research has also demonstrated the use of Unity together with ROS as digital and communication components for robotic arm digital twin applications in smart manufacturing. Introducing Unity in the course therefore gave students experience with software that is not limited to game development, but is also relevant for modern engineering workflows involving simulation, automation, and digital system representation.

4.2 Web Interface for battery module disassembly

The second implementation interface was developed as a browser-accessible 3D factory simulation. It was designed to connect sensor input with a practical manufacturing scenario. In contrast to the Unity Interface, which mainly introduced sensor-driven interaction in a local simulation, the Web Interface extended the platform toward shared interaction, production planning, and process-oriented decision making. The simulation represents a battery disassembly factory in which students control a mobile robot inside a virtual production hall and move battery cells between different processing stations. Figure 4 shows the two browser modes used in the Web Interface: the player mode, in which students navigate the robot through the factory using their sensor system, and the viewer mode, which provides a top-down overview for supervision, route comparison, and live monitoring.

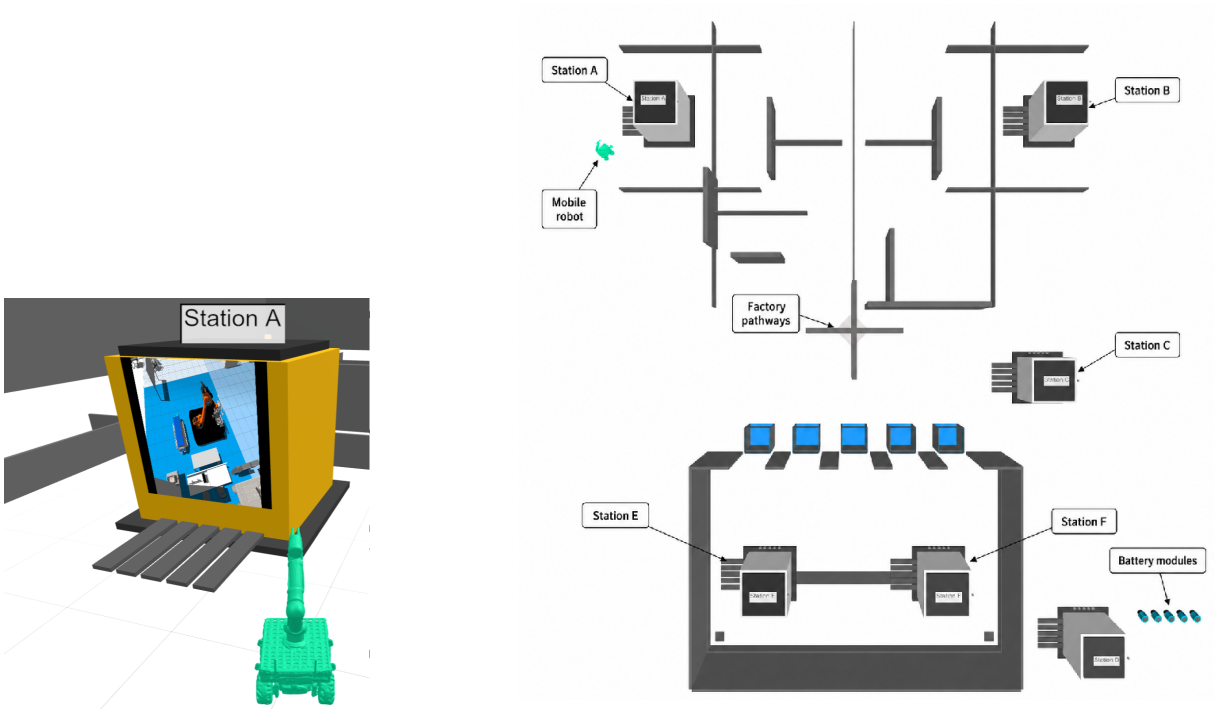
The factory was implemented as part of the Web Interface using Three.js. The virtual production hall includes textured floors, walls, obstacles, a maze-like layout, an upper platform, elevators, conveyor belts, processing machines, robot models, and visual feedback from the stations. Six processing stations, labeled A to F, are distributed across the ground floor and the upper level. The Web Interface therefore gives students a simplified but process related representation of battery handling, station interaction, and robotic movement inside a factory layout.

At the beginning of the activity, each group assigns their sensor system from the start menu. This device forms the connection between the physical sensor system and the virtual factory. The robot is then controlled through sensor data. Different physical inputs are mapped to different factory interactions.

The main task in the simulation is the handling of battery cells. Students find and pick up a battery cell, transport it to a processing station, place it on the station conveyor, complete the station task, retrieve the processed part, and continue through a sequence of stations. A battery run is evaluated after the part has passed through three stations. This gives the Web Interface a clear process objective rather than simple robot driving.

A central educational feature of the Web Interface is the comparison between the selected route and the optimal route. The optimal path is not fixed for all groups. Instead, the system generates group specific timing configurations using station data such as base processing time, probability of error or rework, rework duration, and expected total processing time. The simulation can therefore compare the route chosen by the students with the best route expected for their group. A route may appear short in the 3D factory layout but still be less efficient because of station processing times, rework probabilities, or process delays.

The Web Interface also supports multiplayer operation. Several robots can exist in the same factory session, while their positions, directions, colors, carried parts, and run progress are synchronized live. This enables



(a) (b)

Figure 4. Web Interface for battery module disassembly: (a) player mode for student navigation of the mobile robot with the sensor system in the shared 3D factory; (b) viewer mode with a top-down factory overview for instructor supervision, live monitoring, and route comparison.

multiple student groups to work in parallel inside the same shared factory interface. As a result, the browser application can react immediately to changes from hardware inputs, other players, or spectator views.

In addition to the student player mode shown in Figure 4(a), the system includes a viewer mode for instructors and observers, shown in Figure 4(b). This mode provides a top-down view of the factory and can display live information about active groups, station progress, selected routes, optimal route comparison, trial results, and best runs. The viewer interface therefore supports teaching, supervision, and discussion during the workshop. It allows instructors to monitor several groups at the same time and use the live factory state as a basis for explaining routing decisions, process uncertainty, embedded input, and production efficiency.

From an educational perspective, the Web Interface combines several engineering topics in one accessible system. Students work with sensor hardware, real-time data exchange, browser-based 3D visualization, robot-style navigation, station interaction, battery disassembly logic, route planning, expected processing time, and multiplayer synchronization. This makes the Web Interface an interactive learning setting in which students can see how physical sensor inputs affect a shared digital production process.

Overall, the Web Interface forms the second stage of the proposed platform. It builds on the sensor acquisition and communication principles introduced in the Unity Interface, but extends them toward a scalable shared factory scenario with industrial relevance. The battery disassembly context also exposes students to practical questions related to automation, process planning, and robotic involvement in battery module handling. This prepares students for the final implementation stage, where similar sensing and communication principles are connected to advanced robotic simulations and selected physical robot systems.

4.3 ROS Interface for robotic simulation and physical robot extension

The ROS Interface represents the most advanced stage of the platform. It extends the sensor interaction concept from the Unity Interface and Web Interface to robotics simulation and physical robotic systems. In contrast to

the Unity Interface and Web Interface, this stage introduces students to robotics software workflows in which sensor data are not only used to trigger visual feedback but are integrated into robot control and motion planning pipelines. Figure 5 shows the ROS Interface from sensor hardware and wireless data transfer to ROS 2 topics, student control nodes, simulation-first robot experiments, and selected transfer to physical robots.

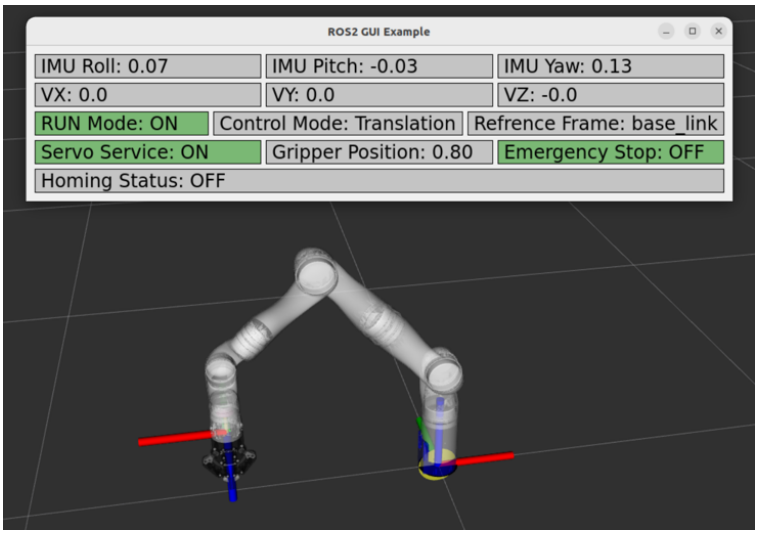


Figure 5. ROS Interface connecting student sensor hardware to ROS 2 simulations and physical robot transfer through containerized workspaces.

Two ROS 2 repository workspaces were prepared for the ROS Interface and provided to the students. Both were distributed as containerized workspaces so that students could work with robotics software without configuring complex dependencies manually or requiring direct access to physical robots. The repositories were explicitly intended for simulation use and did not require a robot IP address, hardware drivers, or connection to the real systems. The workspaces included robot visualization, ROS 2 control with fake hardware, MoveIt 2 for motion planning, and Gazebo or Ignition for 3D simulation. This allowed students to receive sensor data from external microcontrollers, transmit the data to ROS 2 over Wi-Fi, map the received values to robot commands, and test the resulting behavior in a safe simulated environment.

Within the ROS Interface, the workspace addressed different robotic scenarios. The Kinova Gen3 setup focused on robotic arm control, where sensor values could be mapped to joint targets, trajectory commands, or motion planning tasks. It therefore introduced students to manipulator control, joint states, command interfaces, and trajectory execution. The Spot setup extended the concept to a mobile robotic system with a simulated mobile base and arm, where students could test navigation and motion commands in a more complex robot configuration. Together, the two ROS 2 simulations showed that the same sensor interaction chain can be transferred to different robotic platforms, from stationary manipulation to mobile robotics.

A key feature of the ROS Interface is that the robotic workspace was designed as a simulation-first workspace, where students can explore robot control concepts without the risk, supervision requirements, and safety constraints associated with direct access to a physical robot. The simulated system allows students to test whether their sensor signals are received correctly, whether the mapping from sensor values to robot commands is reasonable, and whether the resulting robot behavior matches the intended interaction logic.

The use of ROS 2 also changes the technical depth of the project. Students are introduced to robotics middleware, topics, controllers, robot descriptions, joint states, motion planning, visualization, and simulated hardware interfaces. Sensor data can be published to ROS 2 topics, processed by student developed nodes, and converted into robot commands. This creates a clear connection between the sensor system developed in the earlier stages and the software structures used in modern robotic applications.

Docker was used to make the ROS Interface more reproducible and easier to distribute. Since ROS 2 installations and robotic software dependencies can be difficult to configure on different computers, the containerized setup reduces setup barriers and helps ensure that student groups work with comparable software environments. This is particularly relevant in a teaching context where several groups must install and run the same robotics stack within a limited project time.

The ROS Interface was also designed as a transition point toward physical robotic systems. At WZL institute, both a Kinova robotic arm and a Spot robot were available as real hardware platforms. After the simulation stage, selected high performing groups were offered the opportunity to transfer their sensor driven control concepts to the physical robots. These groups successfully connected their developed sensor systems to the real robot, showing that the ROS simulations were not isolated demonstrations, but preparatory stages for real robotic interaction.

From an educational perspective, the ROS Interface provides the strongest link to advanced robotics and automation. Students move from controlling a virtual character in Unity, to interacting with a shared Web Interface factory, and finally to using their sensor systems within ROS 2 robotics simulations. The physical robot extension adds an additional level of authenticity for selected groups, while the simulation first structure keeps the ROS Interface scalable and safe for the full course. This makes the ROS Interface the highest complexity implementation within the proposed platform and demonstrates how the same sensor interaction chain can be extended from accessible digital interfaces to realistic robotic workflows.

4.4 Comparison of the three implementation interfaces

The three implementation interfaces differ in technical depth, scalability, and educational focus, but they follow the same sensor interaction principle. In all stages, students acquire physical sensor data, process the signals, transmit the interpreted values, and use them to influence the behavior of an external system. The main difference between the interfaces is the complexity of the system response and the type of engineering context in which the sensor data are used.

This progression allows the platform to support different teaching depths within the same course concept. The Unity Interface emphasizes intuitive understanding and early testing. The Web Interface introduces distributed interaction and process oriented decision making. The ROS Interface exposes students to robotics software workflows, motion planning, simulated hardware interfaces, and the transition from simulation to real robotic platforms. As a result, the platform does not depend on one fixed laboratory setup, but provides a flexible sequence of interfaces that can be adapted to available time, student numbers, laboratory access, and course objectives.

Together, the interfaces create a continuous learning path from sensor-driven interaction to industrial process simulation and advanced robotic application. This structure preserves the practical value of working with real sensor hardware while making the system scalable through digital and simulation environments.

5. STUDENT IMPLEMENTATIONS

This section summarizes the student implementations at the level of course deployments, group outcomes, and recurring technical observations. Since the projects were completed as part of teaching activities, the discussion focuses on aggregated results and anonymized examples of student work rather than on identifying individual groups. This makes it possible to describe the practical value of the platform without depending on permission to publish detailed implementations from specific students.

5.1 Course deployment and project workflow

The platform was deployed in two semesters of the RWTH master's course "Robotic Sensor Systems" and in one international summer school. The first course deployment started in the winter semester 2024/2025 and used the Unity Interface. The following course deployment was in the winter semester 2025/2026 and used the ROS Interface. The Web Interface was applied during a summer school with a partner university in Italy in 2025. In total, the two course deployments involved approximately 200 students, while the summer school added around 25 students working in smaller project groups.

At the beginning of each course semester, the project goals and the graded task list were introduced to the students. A registration period allowed students to form groups of three to five members. Students who registered individually could be assigned to groups afterwards, which ensured that the project remained accessible even when students did not already have a team. Each group received a sensor component package that could be used at home, during tutorials, or during the organized work sessions. The package included an ESP32 microcontroller, the touch sensing capability of the ESP32 pins, an MPU6050 module with gyroscope, accelerometer, and temperature signals, an infrared sensor, a rotary encoder, breadboards, and jumper wires. Figure 6 shows the component set distributed to the groups.

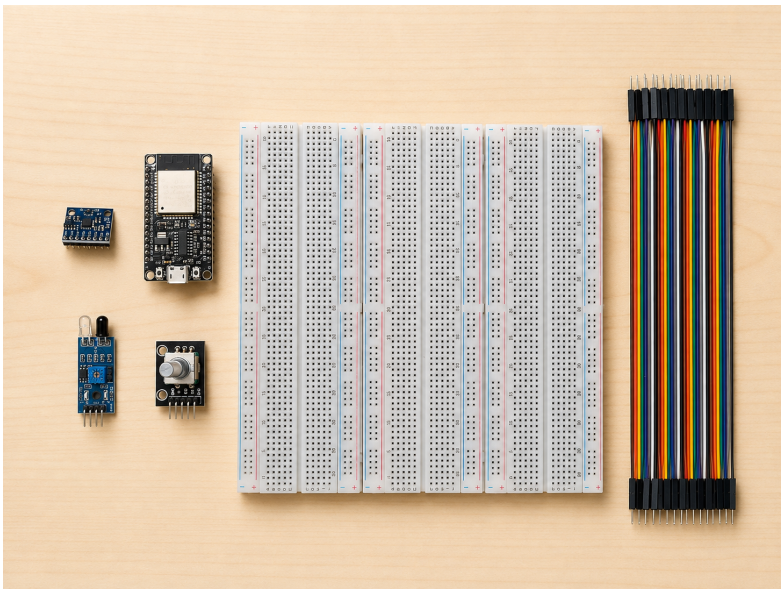


Figure 6. Sensor component package used by student groups for the project implementation.

The project work was integrated with the lecture and tutorial structure of the course. Theoretical background on the individual sensors, signal acquisition, and communication concepts was introduced throughout the semester while groups worked independently on their implementations. In both course semesters, two optional supervised work sessions were offered, one in the middle of the semester and one close to the final evaluation. Each work session provided approximately eight hours of dedicated project time. They were not mandatory, but they created a productive setting in which groups could test hardware, compare implementation approaches, ask questions, and exchange solutions. Figure 7 shows a photograph taken during one of the organized work sessions.

At the end of each course semester, a full evaluation day was organized. Each group first gave a five minute presentation describing its implementation concept, sensor selection, challenges, and final solution. The presentation was followed by a live demonstration and a short discussion. At the end of the evaluation, the sensor components were returned so that the same hardware packages could be reused in future course offerings.

5.2 Assignment requirements in the three deployments

The Unity Interface deployment used an upfront graded task list that required full control of the robot character, including regular movement, running, and jumping. It also required full camera control, door control, display of the temperature, filtering of sensor signals, a creativity component, and a five minute presentation. The assignment of specific sensors to these functions was intentionally left open. This design decision was important because it forced each group to analyze the available sensors, decide which measured quantities were suitable for which interaction, and develop an individual mapping strategy. As a result, the solutions differed even though all groups worked with the same basic hardware package and Unity scene.

The ROS Interface deployment followed the same general project structure but increased the technical depth of the assignment. The graded tasks required a working ROS 2 system that could receive sensor data over Wi-Fi



Figure 7. Photograph from an organized work session in which student groups tested hardware, discussed implementation approaches, and received supervised support.

using TCP, UDP, ROS 2 communication, or MQTT. Students also had to publish sensor data to appropriate ROS 2 topics, map sensor inputs to robot control commands, visualize the robotic arm in RViz2, document the implementation in a Git repository, present a live demo, include a creativity component, and give a five minute presentation. Compared with the Unity Interface deployment, the main difference was that students had to connect the sensor system to a robotics software workflow rather than to a local game scene.

The summer school deployment used the predeveloped Web Interface in a more compact workshop format. Student groups connected the same sensors to the shared factory layout and used them to move a robot and pick and place battery modules through disassembly stations in a required order. The station order was based on group-specific calculations, which connected the sensor interaction task to process planning and decision making. Since all groups were synchronized with the same Web Interface factory scene, the task also emphasized shared interaction, timing, and coordination.

5.3 Observed outcomes and project quality

The final demonstrations showed that the platform enabled a broad range of successful solutions while still preserving a common learning structure. With the Unity Interface, the visual feedback of the game scene made the sensor system behavior understandable. Many groups were able to move beyond simple threshold-based inputs and implemented smoother interaction concepts using filtering, scaling, or combined sensor states. The open assignment of sensors to functions also led to different control concepts, for example using IMU motion for character movement, rotary encoder input for camera rotation, infrared sensing for interaction events, or touch inputs for discrete actions.

In the ROS Interface, groups achieved stable wireless communication, published structured sensor data to ROS 2 topics, mapped the sensor values to robot motion in a controlled way, and demonstrated the result in RViz2 or simulation. After the evaluation session, the semester concluded with a voluntary one-day workshop for the four best-performing groups, selected according to their results on the graded task list. In this workshop, the

groups worked with two physical Kinova robotic arms and transferred their simulation-based solutions toward real robot control for a pick-and-place task. This activity was not part of the graded assignment, but all four invited groups attended, which indicates a high level of student motivation. All four groups were able to control a Kinova arm during the work session. This outcome showed that the simulation stage provided useful preparation for real robotic interaction and that selected students could transfer the same sensor interaction chain to physical robot hardware.

The summer school demonstrated that the Web Interface can support a shorter and more collaborative teaching format. The shared factory scene encouraged students to compare approaches, discuss routing strategies, and quickly observe the consequences of their sensor mappings. Even within the limited workshop time, the groups were able to connect the physical sensors to a process-oriented digital task.

Across all deployments, the creativity component was an important driver of student motivation. Several groups outperformed the minimum requirements by adding additional components to the project. Examples included the design of custom computer-aided design (CAD) housings and 3D printed cases that made the sensor system easier to hold and operate, the use of neural-network-based gesture recognition on the microcontroller to identify additional IMU-based actions, and improvements to the Unity Interface to make the scene more realistic and engaging. These additions showed that the platform supported not only task completion, but also student ownership and engineering creativity.

5.4 Observed implementation challenges

The main challenges differed between the three interfaces. Across all deployments, students had to solve problems in hardware connection, signal processing, communication, and application-level mapping. However, the dominant source of difficulty shifted as the interface complexity increased. In the Unity Interface deployment, many challenges were related to wiring and testing the hardware, reducing noisy readings, and transforming raw values into stable game commands. In the Web Interface deployment, the main difficulties were communication reliability, synchronization with the shared factory interface, and mapping sensor inputs to the required station sequence. In the ROS Interface deployment, the challenging part was the software integration: receiving sensor data reliably, publishing it to topics, connecting it to robot control, and debugging the interaction across the microcontroller, communication module, ROS 2 nodes, and simulation environment.

The work sessions were especially useful because the challenges were distributed unevenly across groups. One group might solve a communication issue early, while another group might find a better filtering strategy or a more practical sensor mapping. The shared sessions therefore helped turn individual troubleshooting into peer learning. This observation is relevant for future deployments: even when the platform is designed for independent group work, scheduled collaborative support periods remain important for maintaining progress and reducing frustration.

6. EDUCATIONAL EVALUATION

Student perceptions were collected using an anonymous post-project questionnaire. Because the questionnaire was administered after the project had ended, it was difficult to reach students again and obtain a high response rate. The exploratory analysis therefore included $n = 11$ analyzed responses, and the results should be interpreted as indicative student feedback rather than as a statistically representative evaluation. The instrument covered five dimensions: learning outcomes, collaboration, usability and technical quality, motivation and engagement, and self-assessed skills. Agreement-based items used a five-point Likert scale ranging from 1 = strongly disagree to 5 = strongly agree. Self-assessment items used a parallel five-point confidence scale ranging from 1 = not confident at all to 5 = very confident. One attention-check item was included in the learning-outcomes section; this item was used only to verify attentive participation and was excluded from the calculation of the learning-outcomes score.

The respondent profile is summarized in Figure 8. The profile includes academic background, time spent on project work outside scheduled sessions, prior sensor familiarity, and prior programming familiarity. This information provides context for interpreting the questionnaire results, especially because the sample size is

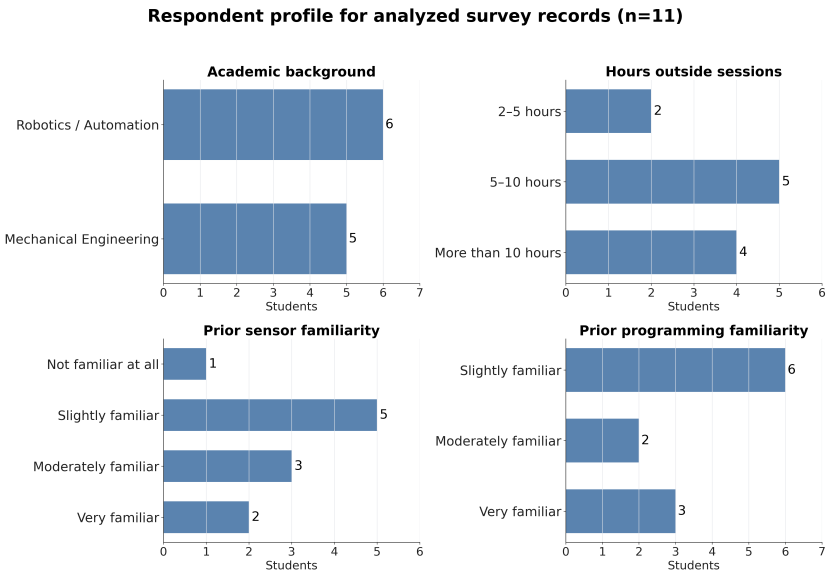


Figure 8. Respondent profile of the analyzed questionnaire sample, including academic background, time spent on project work outside scheduled sessions, prior sensor familiarity, and prior programming familiarity.

small and the evaluation should therefore be understood as exploratory rather than as a basis for statistical generalization.

The student evaluation results are shown in Figure 9. Overall, students evaluated the project positively across all measured dimensions. Motivation and engagement received the highest score, with a mean of 4.72/5 and 96.3% positive item responses. Learning outcomes were also rated highly, with a mean of 4.44/5 and 94.5% positive responses. Positive responses were defined as ratings of 4 or 5 on the corresponding five-point scale. The most mixed responses were observed for usability and technical quality, particularly regarding whether technical problems interrupted learning. These responses suggest that students perceived the project as engaging and educationally valuable, while also indicating that technical robustness remains an area for improvement.

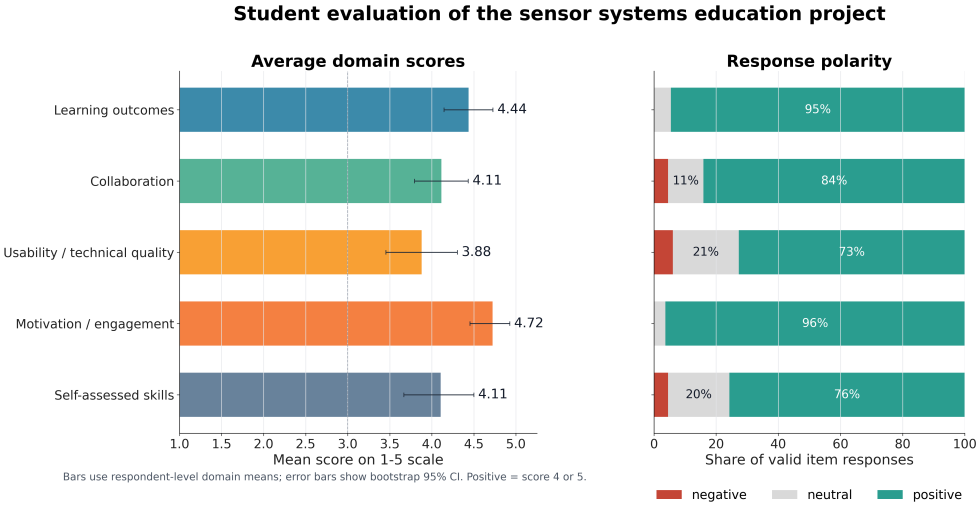


Figure 9. Student evaluation results across the five questionnaire domains: learning outcomes, collaboration, usability and technical quality, motivation and engagement, and self-assessed skills.

7. CONCLUSION AND FUTURE WORK

This work presented a multi-interface platform for teaching sensor systems through a progressive combination of the Unity Interface, the Web Interface, and the ROS Interface. The platform connects physical sensor acquisition, processing, communication, and external system response within one continuous learning workflow. In this way, students do not only observe isolated sensor values, but experience how measured signals can be interpreted and used to influence the behavior of technical systems.

The platform was implemented through three complementary interfaces. The Unity Interface provided a starting point for sensor interaction in a safe visual scene. The Web Interface extended the concept toward shared interaction, process-related decision making, and distributed control in a digital production scenario. The ROS Interface introduced students to ROS 2 workflows, robotic simulation, and selected transfer to physical robotic systems. Together, these interfaces formed a scalable teaching structure that can be adapted to different levels of complexity, available resources, student experience, and course objectives.

The course implementation showed that real sensor hardware can be combined with digital interfaces in a way that preserves practical engineering experience while reducing the dependence on limited laboratory equipment. Students were able to work with sensor wiring, microcontroller programming, signal interpretation, communication, visualization, and system response within a unified project structure. The use of scalable digital interfaces also allowed many student groups to work in parallel while still engaging with realistic engineering tasks. In addition, the optional robotic extension demonstrated that the same sensor interaction chain can be transferred from the ROS 2 simulation to more advanced robotic applications.

Future work will focus on further extending the platform and improving its evaluation. First, the educational impact of the different implementation stages should be investigated in more detail using larger questionnaire datasets, structured interviews, and comparison with previous course formats. Second, the Web Interface can be expanded with additional process scenarios, more complex station behavior, and a more detailed feedback on route planning, timing, and production efficiency. Third, the ROS Interface can be further developed by increasing the number of simulation tasks, improving the connection between sensor data and robot motion planning, and allowing more student groups to test their solutions on physical robotic systems under controlled safety conditions.

Further extensions may also include additional sensor modalities, camera input, simple machine learning tasks, sensor fusion, and more advanced communication protocols. These additions would allow the platform to support a broader range of teaching objectives, from basic sensor acquisition to intelligent automation and robotic control. Overall, the presented platform provides a flexible foundation for practical sensor systems education and shows how the Unity Interface, the Web Interface, and the ROS Interface can be integrated into one engineering learning platform.

REFERENCES

[1] Prince, M. J., "Does active learning work? a review of the research," *Journal of Engineering Education* **93**(3), 223–231 (2004).

[2] Heradio, R., de la Torre, L., Galan, D., Cabrerizo, F. J., Viedma, E. H., and Dormido, S., "Virtual and remote labs in education: A bibliometric analysis," *Computers & Education* **98**, 14–38 (2016).

[3] Li, J. and Liang, W., "Effectiveness of virtual laboratory in engineering education: A meta-analysis," *PLOS ONE* **19**(12), e0316269 (2024).

[4] Wahyudi, M. N. A., Budiyanto, C. W., Widiastuti, I., Hatta, P., and bin Bakar, M. S., "Understanding virtual laboratories in engineering education: A systematic literature review," *International Journal of Pedagogy and Teacher Education* **7**(2) (2023). https://jurnal.uns.ac.id/ijpte/article/view/85271.

[5] den Beemt, A. V. et al., "Remote laboratories in higher engineering education: Engaging students through active learning," *Education Sciences* **12**(8), 543 (2022).

[6] Jacko, P., Bereš, M., Kováčová, I., Molnár, J., Vince, T., Dziak, J., Fecko, B., Šimon Gans, and Kováč, D., "Remote iot education laboratory for microcontrollers based on the stm32 chips," *Sensors* **22**(4), 1440 (2022).

[7] Sell, R., Czekalski, P., Tokarz, K., Kuaban, G. S., Nikitenko, A., Berkolds, K., and Lipka, L., "An international hands-on iot education platform towards industry 5.0," *Proceedings of the Estonian Academy of Sciences* **74**(2), 198–204 (2025).

[8] Terkaj, W., Kleine, K., and Kuts, V., "Virtual labs for higher education in industrial engineering," *Proceedings of the Estonian Academy of Sciences* **73**(2), 142–149 (2024).

[9] Singh, M., Kapukotuwa, J., Gouveia, E. L. S., Fuenmayor, E., Qiao, Y., Murry, N., and Devine, D., "Unity and ros as a digital and communication layer for digital twin application: Case study of robotic arm in a smart manufacturing cell," *Sensors* **24**(17), 5680 (2024).

[10] Pantusin, F. J., Ortiz, J. S., Carvajal, C. P., Andaluz, V. H., Yar, L. G., Roberti, F., and Gandolfo, D., "Digital twin integration for active learning in robotic manipulator control within engineering 4.0," *Symmetry* **17**(10), 1638 (2025).

[11] Kurtz, M., Benabbou, A., Pons, C., and Broisin, J., "Collaboration in virtual and remote laboratories for education: A systematic literature review," *International Journal of Computer-Supported Collaborative Learning* **20**(4), 549–603 (2025).