Proceedings of the 10th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'24)
Barcelona, Spain - August 19 - 21, 2024
Paper No. CIST 176
DOI: 10.11159/cist24.176

CAN-Bus Remote Laboratory on WebLab-Deusto System

Oleksandr Velihorskyi[1,2], Roustiam Chakirov[1], Christoph Mauel[1]
[1]Bonn-Rhein-Sieg University of Applied Sciences
Grantham Allee 20, Sankt Augustin, Germany
oleksandr.velihorskyi@h-brs.de; roustiam.chakirov@h-brs.de; christoph.mauel@h-brs.de
[2]Chernihiv Polytechnic National University
Shevchenko Str. 95, Chernihiv, Ukraine

Abstract – Technologies of online education, such as videoconferencing, online learning management systems, and remote laboratories play crucial role in modern world, providing equal possibilities for students all over the world. The goal of the paper is to present the results of the development of a remote laboratory for online access to the CAN-Bus experiment for students of electrical engineering programs. The developed remote laboratory setup is based on Remote Laboratory Management System WebLab-Deusto, and extends existing in Bonn-Rhein-Sieg University of Applied Sciences CAN-Bus experiment. The laboratory consists of several layers, which are described in detail in the paper – layer of management system (server with WebLab-Deusto instance), an experiment server based on Raspberry Pi, input-output infrastructure for the connection with the experiment, and finally, the existing experiment equipment. The Python-based web framework Flask and library Weblablib were used in the experiment server, providing a remote laboratory web application and human-machine interface for the interaction with the equipment. The laboratory can be used in remote and on-site modes and can be further integrated to the university's learning management system Moodle.

***Keywords*:** Remote Laboratory, Online Education, WebLab-Deusto, Electronics, Internet of Things

1. Introduction

Since the COVID-19 outbreak, online mode has become one of the essential parts of the learning process in higher education. In the beginning, services for videoconferencing (Zoom, MS Teams, Webex) and learning management systems (Moodle, Whiteboard) were widely spread for lectures and practice lessons and became the first set of tools that were used for continuing educational services in the pandemic world. Such a set of tools was used for communication between academic staff and students, presentations, tests and assessments. After the pandemic was over, the interest in various aspects of online and remote tools for education services declined, but did not perish. Wars in Ukraine and other countries all over the world destroyed university campuses and schools, raising the importance of remote technologies for higher education again. Classical technologies, like services for video conferences, learning management systems, and specialized software for calculation or simulation, are well suited for obtaining by students only some of the most important for further labor markets, such as math-oriented or soft skills. Beside it, especially in the field of engineering, practical-oriented skills are compulsory for successful alumnus employment. Such skills can be achieved only by hands-on experience, work in a laboratory with measurement equipment, specially designed test benches, setups or educational sets [1]. One of such engineering fields is electrical engineering, where students need to have practical experience working with various electronic circuits, developing their own prototypes, and implementation of various communication interfaces.

There are a lot of attempts of "remotizing", or developing remote laboratories, that will help the students of electrical engineering specialties obtain practical experience, since there is no standard approach to remotization of existing laboratories. Zapata Rivera and Larrondo-Petrie proposed UML models for remote laboratories and user roles and interactions with laboratories [2] that should help with the standardization. Many papers describe remote laboratories focused on the programming of microcontrollers or single-board PCs: 32-bit ARM from NXP Semiconductors [3], Raspberry Pi [4]. Beside microcontrollers, a lot of attention has been devoted in recent findings to FPGA-based remote laboratories [6], [7]. It should also be mentioned, that developed remote laboratories can be successfully integrated with existing learning management systems, such as Moodle [5], which can significantly extend the functional of remote laboratories. At the same time, such integration is still at the stage of early experiments and waits for a convenient and unified solution. It should also

be noted, that, as of now, there is a lack of remote laboratories, in which the focus is oriented on competencies in the use of industrial and automotive interfaces, such as CAN-Bus or EtherCAT.

A significant boost in standardization and unification of approaches in the development of remote laboratories brought WebLab-Deusto – Remote Learning Management System (RLMS) [8], which provides a framework for the development of high-level systems [9], as well as a Python library and set of examples for the implementation of remote laboratories at the experiment level [10] since 2006. In 2024, LabDiscoveryEngine, a successor to WebLab-Deusto, should bring new possibilities for the development of Remote laboratories [11].

The paper proposes the results of the development of a remote laboratory for obtaining of practical competencies working with industrial electrical interfaces for the automotive industry, based on CAN-bus. In Chapter 2, the existing non-remote CAN-Bus experiment is explained, and main tasks required for the successful achieving of the planned student's learning outcomes are discussed. Chapter 3 presents and discusses the results of remotization of such an experiment by utilizing RLMS WebLab-Deusto. Conclusion summarize obtained results and raise issues for further consideration and development.

2. CAN-bus experiment hardware and description of the experiment

A functional block diagram of existing CAN-bus experiment hardware, which was used as a basis for the development of remote experiment, is shown in Fig. 1. In total, three sensors, which measured various parameters in the automotive system, were used in the experiment. *The temperature sensor* TMP36 sends the temperature values, and it represents the temperature inside the engine. *The wheel sensor* LWS3 (Bosch) can measure wheel static and dynamic parameters: wheel position and wheel rotation acceleration. This sensor is based on the "anisotropic magneto-resistive" (AMR) principle, where the electrical resistance depends on the direction of an external magnetic field. Finally, *the throttle pedal sensor* (produced by Bosch) measures the position of the throttle pedal. The system also contains one actuator – *the throttle valve* (also produced by Bosch). The position of the throttle valve depends on the information received about the position of the throttle pedal. If the system is configured properly, the idle position of the throttle pedal should correspond to a closed throttle valve, and the full throttle position should correspond to a fully open throttle valve, respectively. Throttle pedal and throttle valve, used in the experiment, are real parts from a Mercedes-Benz auto. An additional display, the *NeoPixel Ring*, is used for the visualization of the steering wheel position.

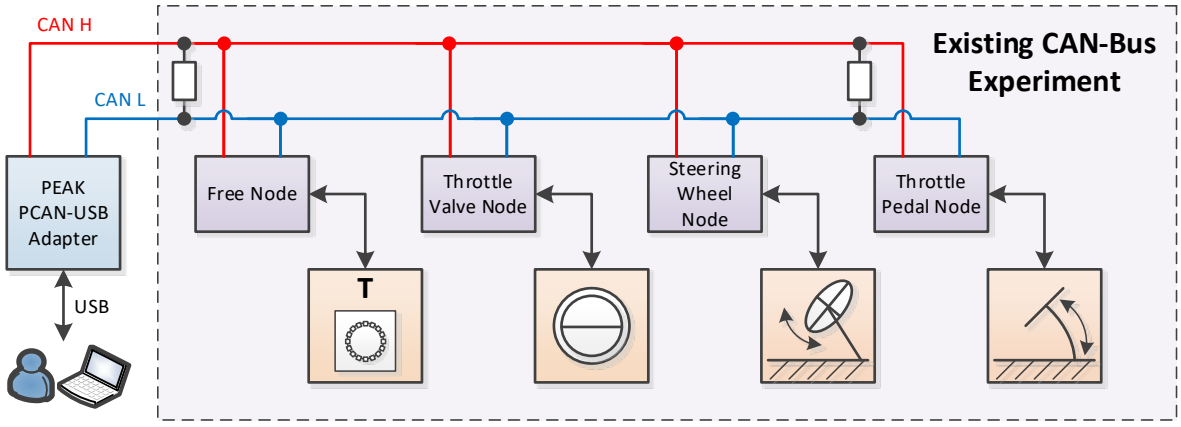


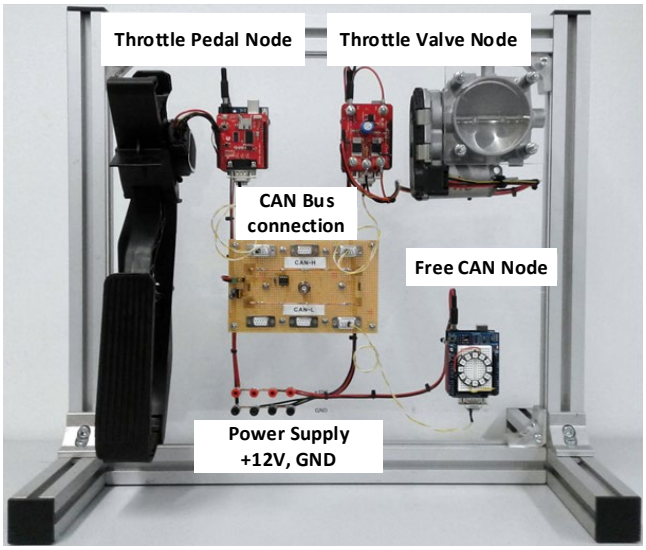
Fig. 1: Functional block diagram of CAN-Bus experiment.

To organize communication between all mentioned above sensors, actuator and display, they are organized in an "automotive" CAN-Bus network, consisting of four nodes. Three nodes are built on the Arduino Uno boards equipped with the CAN-Bus shields: "*Free Node*" with display and temperature sensor, "*Throttle valve node*" with actuator, and "*Throttle Pedal node*" with respective sensor. One additional node, the "*Steering wheel node*" is based on an industrial sensor with an in-built CAN interface, it can be connected additionally to a CAN bus connection module. The general

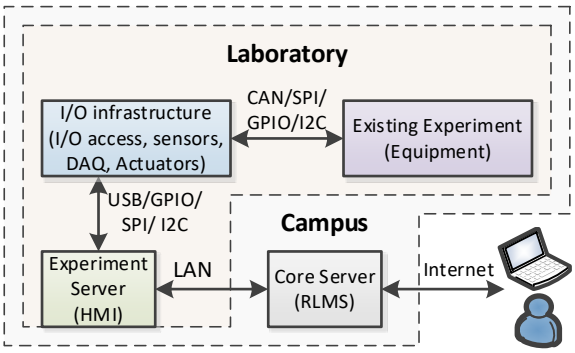
view of the experiment is shown in Fig. 2 (taking into account that "Steering wheel" is implemented as an additional block, it is not shown in the figure). IDs of the network's CAN nodes as well as message formats in the developed CAN-bus are described in Table 1. Students can interact with the CAN-Bus utilizing the PEAK PCAN-USB adapter and PCAN-View software for their personal computer/laptop. Such a set provide the possibility to read messages from the bus and send their own messages in order to configure actuators and sensors (e.g., idle position and full throttle for throttle pedal, zero position for the steering wheel, etc.). Additionally, it is possible to manually connect and disconnect nodes from the bus, connect an oscilloscope to visualize the bit-stream of messages, and later analyze its content (CAN ID, length of the message, data bytes, stuff bits, etc.). Such interactions can be done through physical connection/disconnection of corresponding connectors on the "CAN Bus connection", shown in Fig. 2a.

Table 1: Organization of CAN-Bus.

CAN ID	Description	Data/Information
0x001	*Throttle Pedal Calibration* DLC: 2 bytes	Byte0 = 1 => Calibration of Idle position Byte0 = 2 => Calibration of Full Throttle position Byte 1 – doesn't affect
0x004	*Temperature* DLC: 2 bytes Value range: 0…0x3FF => 10 bits	Byte0 => Higher byte of temperature value Byte1 => Lower byte of temperature value Sensor TMP36, 0x3FF => 5 V, 10 mV/°C, shift 0,5 V
0x0A0	*Throttle Pedal Position* DLC: 2 bytes Value range: 0… 0x3FF	Byte0 => Higher byte of pedal position Byte1 => Lower byte of pedal position
0x7C0	*Calibration of steering wheel sensor* CAL_LWS, 2 bytes	Bits 0…3 – command code word Bits 4…14 – LWS-CAN transmit identifier Bit 15 – free, not used
0x3FF	*Steering wheel sensor data* LWS3_transmit	Position, acceleration – data should be configured by means of CAL_LWS message.



(a)



(b)

Fig. 2: View of the CAN-Bus experiment (a), the functional block diagram of a remote laboratory (b).

To fulfill the requirements of the experiment and obtain the required competencies for developing electronic systems equipped with CAN-Bus, students need to implement the following tasks:

1) Connection of three nodes (throttle pedal, throttle valve, and free node) to the CAN-bus, connection of the PEAK PCAN-USB adapter and its configuration via PCAN-View software.
2) Reading in PCAN-View software the bit-stream and decoding the values from the temperature sensor (CAN-ID 0x004).
3) Calibration of throttle pedal sensor for idle and full-throttle positions by means of messages send to CAN-ID 0x001 via PCAN-View software. After the calibration, students should check the data from the sensor (CAN-ID 0x0A0) as well as the quality of the calibration in two pedal positions by observing the throttle valve position (it should be fully open and fully closed for two border positions of the throttle pedal).
4) Connection of steering wheel sensor to CAN-bus, reading the information about current position and speed of rotation of steering wheel in PCAN-View software (CAN-ID 0x3FF).
5) Calibration of zero position of the steering wheel by sending the control message to CAN-ID 0x7C0 and checking the zero position on NeoPixel Ring display.
6) Disconnection of all nodes except the free node, observation and decoding of bit-stream on oscilloscope CAN-bus message from temperature sensor.

So, finally, after successful implementation of all mentioned above tasks, students have the final learning outcomes, such as understanding of communication principle and design of CAN-bus, reading and sending CAN-bus messages, etc.

3. CAN-bus Remote experiment

As described in the previous part of the paper, the educational test bench is focused on the work in the university laboratory. To provide remote access to the experiment, the existing test bench was extended with additional components. The general structure of the developed remote laboratory is shown in Fig. 2b, which consists of:

- The *Core Server*, which provides the authentication of remote users and further access to the experiment server, was implemented in the so-called Remote Laboratory Management System (RLMS).
- The *Experiment Server*, which implements the human-machine interface for the communication of remote users with physical equipment and communication with sensors/actuators, is used in the following layer;
- *I/O infrastructure*, which provides the measurement of data from experiments and control of existing experiment hardware remotely. It can consist of various sensors, actuators, data acquisition components (DAQ, etc.), and depends on the hardware used in the experiment. This layer is an interface between the hardware and software (server) parts of the remote laboratory.
- *Existing experiment equipment*: equipment, tools, and hardware, used in the laboratory for implementation of the experiment. In our case, it is the equipment of the CAN-Bus experiment, shown in Fig. 2a, and described in the previous part of the paper.

Therefore, in general, a remote laboratory has a hierarchical structure and consists of four layers. It should be noted that one core server could control the access for multiple experiment servers, and one experiment server can control various experiments, but in our case, only one experiment was used.

3.1. Core server and RLMS

A Remote Laboratory Management System (RLMS) has been deployed and configured on the Core server. As RLMS, WebLab-Deusto [9] was used in the project, which provides high-level and management functions, including:

- User management (students, teaching staff, working with user profiles, granting access, groups, etc.).
- Limitation of access to the hardware (if the experiment is used by the student, no other student can't access the equipment).
- Parallelization of access: if there are several of the same experiments connected to one remote laboratory, controlled by WebLab-Desuto, the student can use the first free experiment, or, if there are any frees of them,

the student will be added to the queue, and he/she can see the time, that remains before the equipment becomes available.
- Controlling the time of the experiment and automatically finishing it (e.g., disconnecting and switching off all hardware equipment) when the time dedicated to the execution of the experiment was over.
- Collection and representation for the administrator of the remote laboratory top-level statistics of remote access to the equipment, including time, number of students, heat map of the user's locations, etc.

The core server has an IP address accessible from the Internet, there are also the functions of routing and limitation of access only for internal IP addresses of experiment servers (firewall). It means the user can interact only with allowed IP addresses in the local network of the university.

3.2. Experiment server

The layer of the experiment server can be implemented in several ways. The first one is a server PC, but it has limitations on the available interfaces for the connection with the I/O infrastructure layer. The second option is an industrial PC, which usually has a set of interfaces, including access to some GPIO, RS-232/422/485, USB, etc. At the same time, such an option has a lack of peripheral interfaces, such as SPI or I2C that can be used for the development of a cheap but efficient data acquisition system. In addition, Industrial-grade PCs are usually expensive, because of their high protection level or high reliability. The third possible solution for the experiment server is a single-board PC, such as a Raspberry Pi or BeagleBone. Such PCs have enough computational capabilities for tasks of the experiment server, and, that is important, a lot of possibilities for connection (embedded Ethernet, USB, RS-232, I2C, PWM output, etc.), as well as a lot of so-called "hats" with additional interfaces (e.g., CAN, isolated relays, etc.). Taking into account our limitations (the solution should be cheap but efficient), a Raspberry Pi 3 single-board PC was used as an experiment server in the project.

As the experiment server should be a software interface between the experiment and the core server with RLMS, Weblablib [10] was used for the development of the interface with WebLab-Deusto. Weblablib is a Python library for the creation of instances of remote laboratories, controlled by the WebLab-Deusto RLMS. Taking into account that it is based on Python, all other routines, such as communication with the sensors and actuators on the I/O infrastructure layer and implementation of the human-machine interface (HMI) for the user, are also developed on Python. HMI, in our case, is a web application that was developed on the Flask framework [12]. Additional libraries and packages, such as "can", "rpi_hardware_pwm", "RPi.GPIO" were used for the realization of interfaces with the I/O infrastructure layer.

The developed HMI is shown in Fig. 3, which is divided into 5 main zones.

The *"Control"* zone consists of the main elements for controlling the experiment's hardware. The first part of such control is two round sliders for setting the position of the throttle pedal and rotation direction and speed of the steering wheel through two servo motors. Additionally, the user can connect and disconnect from the CAN-Bus of any node by using four "on"/"off" toggle switches. A schematic image of the road above the control block represents the car's movement and corresponds to the position of the throttle pedal and steering wheel. The dividing line on the road moves faster when the throttle valve opens wider, and the road visually bends in the direction following the steering wheel's position.

"CAN messages" zone is used for receiving and visualization, as well as for preparing and sending messages on CAN-Bus. The interface is similar to the PCAN-View software structure, which is helpful for users. The control button "Read CAN message" is used for reading one message from CAN-Bus, this new message is represented by a red color. CAN-ID, length of data in bytes, data bytes can be seen there. A set of received CAN messages can be saved to the text file by clicking the special button. To send the message to CAN-Bus, the user should fill in the fields below (CAN-ID, length, and data) and press the "Send" button. Students can configure the throttle pedal (idle and full throttle positions) and steering wheel (zero position) through that field.

The *"Experiment view"* zone has a toggle switch and a camera view window. Web-camera Logitech C270 was used to show the student real hardware in the lab and monitor its execution, e.g., the movement of the throttle pedal and throttle valve, the position of the steering wheel on display, etc.

The *"Bitstream read"* zone represents functionality similar to the "logical analyzer". Utilizing that part, the student can read the CAN message from the CAN-bus (it is recommended to disconnect all nodes except one) and save it to the image for further interpretation: start bit, arbitration, control, data, and CRC bits.

A list of all interactions with the remote experiment can be seen in the *"Logging"* zone. It includes a timestamp (date and time) and the description of performed actions in chronological sequence. This data can be used for further preparation of the report after the completion of the experiment.

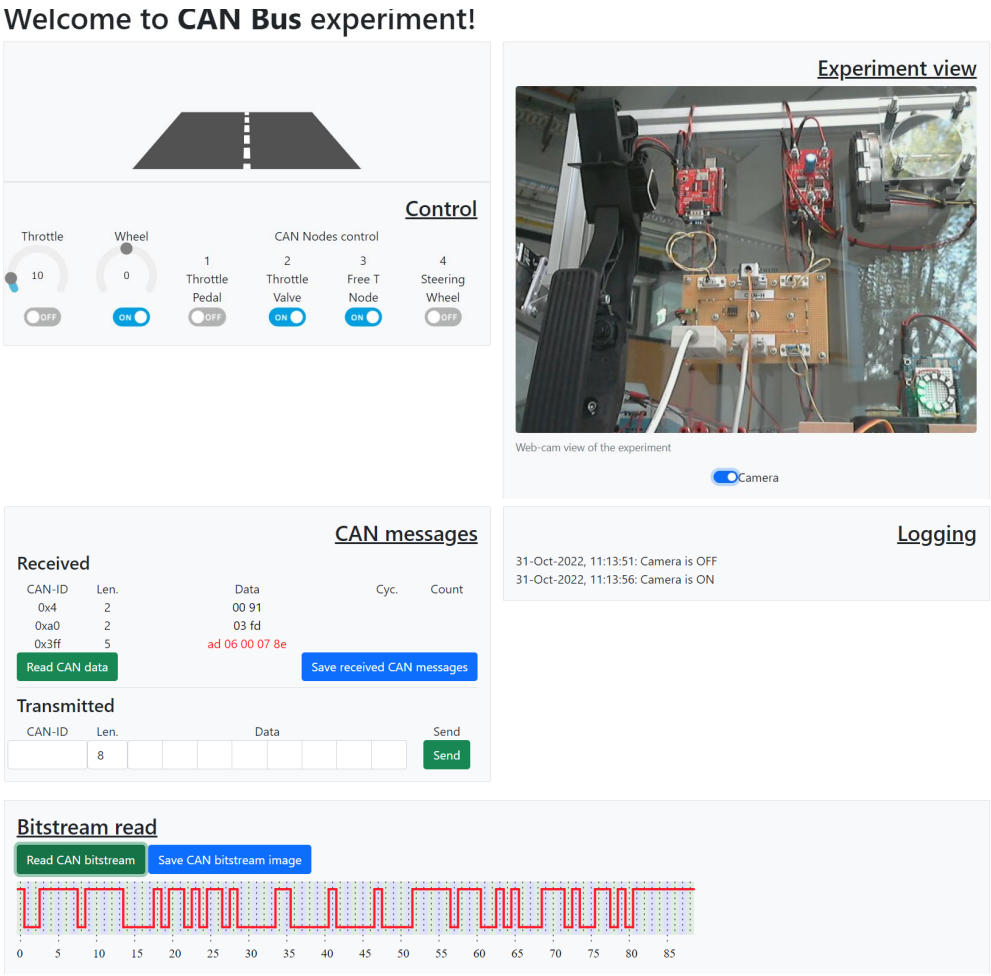


Fig. 3: Flask web-application (HMI) for CAN-Bus remote experiment.

3.3. I/O infrastructure layer

The last, but not the least, part of the remote laboratory consists of specially developed hardware, responsible for the remote interaction with the existing CAN-bus experiment equipment. A functional block diagram of the remote CAN-Bus experiment, including a detailed representation of the existing experiment's equipment and additional infrastructure for interfacing with the experiment server, is shown in Fig. 4. Taking into account that the experiment server is based on a Raspberry Pi and the main task for the interaction is communication with the experiment's CAN-bus, the CAN-SPI Hat "*PiCAN 2 Hat*" was used as the core element for the implementation of that interface instead of the PEAK PCAN-USB adapter. Additional RxD output from CAN-SPI Hat is used for the reading of received messages and their further visualization in the developed HMI (see details in the previous part and Fig. 3). Besides it, Raspberry Pi is also responsible for the on/off control of all CAN-Bus nodes, and an eight-channel relay expansion board was used

for it. Finally, these parts are stacked on each other (Expansion board with DIN-rail holder – PiCAN 2 Hat – Raspberry Pi) and mounted on DIN-rail. AC/DC power supplies with output voltages of +5V (for Raspberry Pi) and +12V (for CAN-Bus nodes, sensors and actuators) are also mounted beside the expansion board on DIN-rail as shown in Fig. 4. Finally, DIN-rail has been connected to the back side of the experiment test bench. The view from the back side of the assembled CAN-Bus remote experiment is shown in Fig. 5.

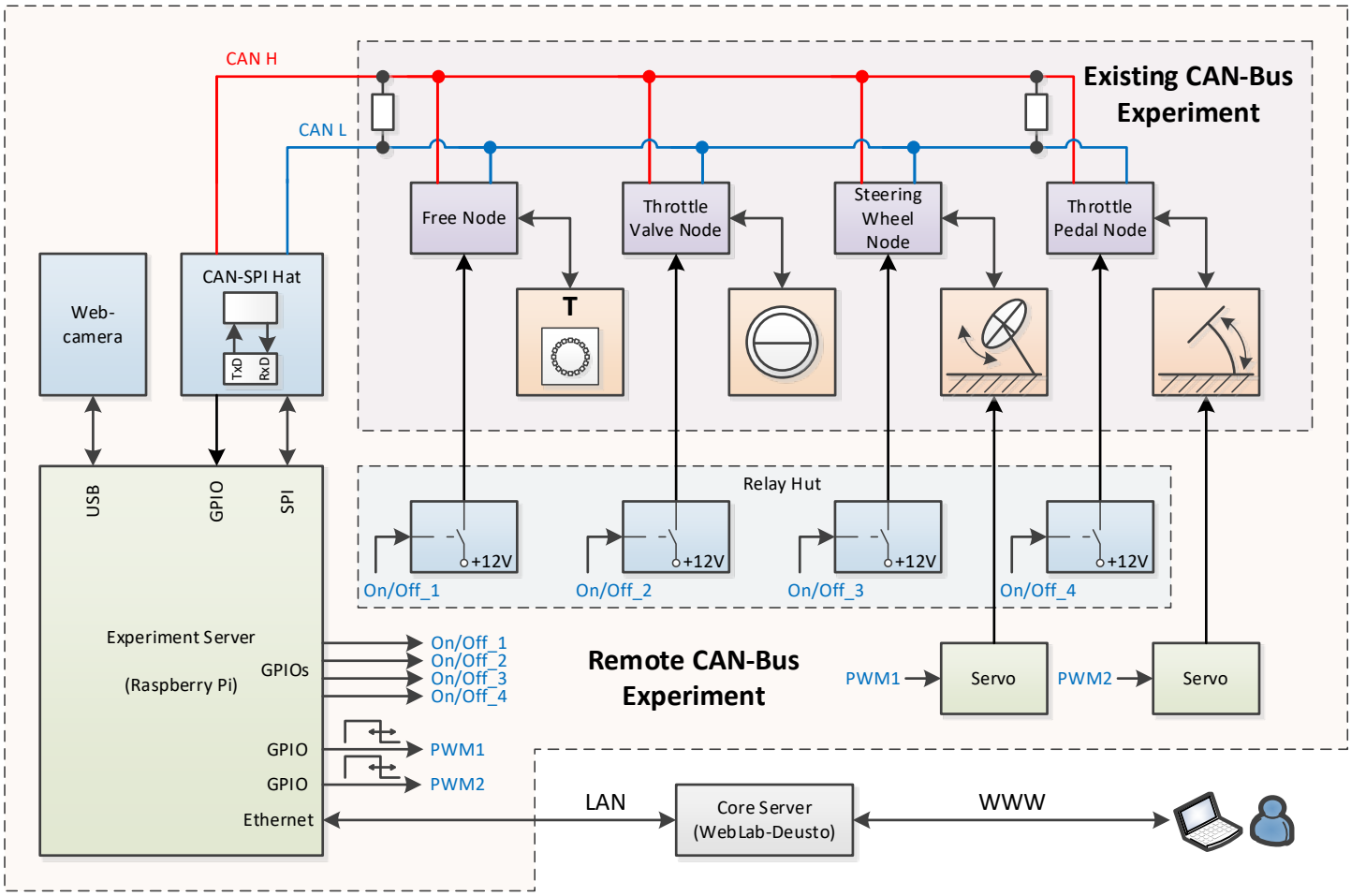


Fig. 4: Functional block diagram of CAN-Bus remote experiment.

4. Conclusion

The presented work shows the result of the development of the Remote Laboratory for obtaining practical competencies in understanding communication principles, transmission and receiving of messages, and design of CAN-Bus. Due to the RLMS WebLab-Deusto, the developed remote laboratory has high-level management functions and can be further embedded in the university learning management system. The developed Remote Laboratory can be used in remote mode as well as be easily transformed to laboratory mode (by disconnection of two servo-drives), which provides flexibility in use. Through the utilization of the Raspberry Pi as an experiment server, existing on-the-market components, such as “hats” and “expansion boards” can be used, which is also useful for replication in other universities. The obtained results can be further extended to the integration of the developed remote laboratory into the university’s learning management system, Moodle.

Acknowledgements

The current work and developed remote experiment were supported by the German Academic Exchange Service (DAAD) and the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung) the program “Ukraine Digital – 2022”, project “FUTURE – Fostering United Teaching in Ukrainian Remote Education”.

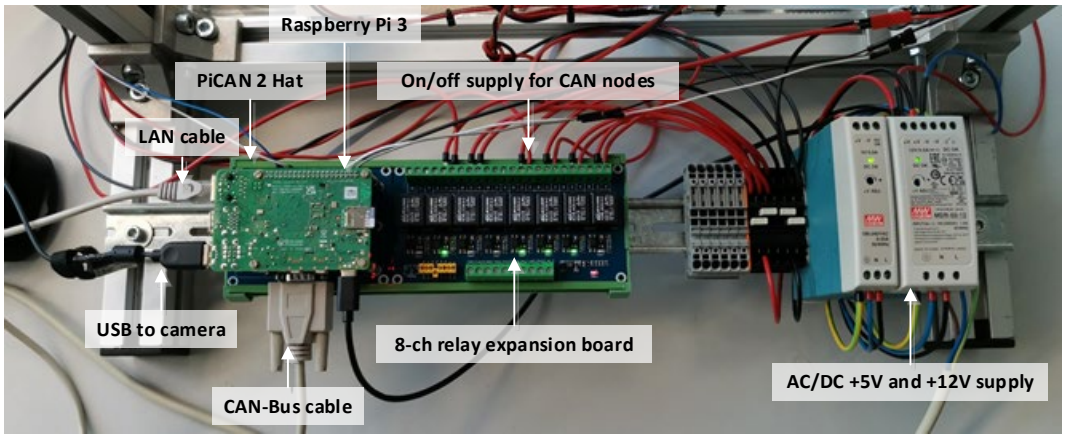


Fig. 5: Assembled experiment server and CAN-Bus remote experiment.

References

[1] D. Satterthwait, “Why are ’hands-on’ science activities so effective for student learning?” *Teaching Science*, vol. 56, no. 2, p. 7–10, 2010.

[2] L. F. Z. Rivera and M. M. Larrondo-Petrie, “Models of remote laboratories and collaborative roles for learning environments,” in *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, Madrid, Spain, 2016, pp. 423-429, https://doi.org/10.1109/REV.2016.7444517

[3] H. S. Jo and R. S. Jo, “Design and Development of Remote Laboratory System to Facilitate Online Learning in Hardware Programming Subjects,” in *2020 13th International UNIMAS Engineering Conference (EnCon)*, Kota Samarahan, Malaysia, 2020, pp. 1-5, https://doi.org/10.1109/EnCon51501.2020.9299326

[4] P. Baizan, A. Macho, M. Blazquez, F. Garcia-Loro, C. Perez, G. Diaz, E. Sancristobal, R. Gil and M. Castro. “IoT Remote Laboratory Based on ARM Device Extension of VISIR Remote Laboratories to Include IoT Support,” in *Cyber-physical Systems and Digital Twins. Lecture Notes in Networks and Systems*, 2019, vol. 80, pp 269–279. Springer, Cham. https://doi.org/10.1007/978-3-030-23162-0_24

[5] T. Alkhaldi, I. Pranata, R.I. Athauda, “A review of contemporary virtual and remote laboratory implementations: observations and findings,” *J. Comput. Educ*. 2016, 3, pp. 329–351. https://doi.org/10.1007/s40692-016-0068-z

[6] V. Baida and S. Ivanets, “Building a Virtual Hardware Laboratory with FPGA and Raspberry Pi Integration,” in *V International Scientific and Practical Conference Theoretical and Applied Aspects of Device Development on Microcontrollers and FPGAs*, Kharkiv, Ukraine, 2023, pp. 36-37. https://doi.org/10.35598/mcfpga.2023.011

[7] A. Magyari, Y. Chen, “FPGA Remote Laboratory Using IoT Approaches,” *Electronics*, 2021, vol. 10, p. 2229. https://doi.org/10.3390/electronics10182229

[8] J. Garcia-Zubia, D. Lopez-de-Ipina, P. Orduna and U. Hernandez-Jayo, “Experience with WebLab-Deusto,” in *2006 IEEE International Symposium on Industrial Electronics*, Montreal, QC, Canada, 2006, pp. 3190-3195, https://doi.org/10.1109/ISIE.2006.296127

[9] WebLab-Deusto (2024, May 30). Scalable, web-based and experiment-agnostic remote laboratory management system [Online]. Available: https://github.com/weblabdeusto/weblabdeusto

[10] Weblablib (2024, May 30). Python library for creating WebLab-Deusto unmanaged laboratories [Online]. Available: https://github.com/weblabdeusto/weblablib

[11] LabDiscoveryEngine (2024, May 30). [Online]. Available: https://labdiscoveryengine.labsland.com/source

[12] Flask (2024, May 30). [Online]. Available: https://flask.palletsprojects.com/