

Web based User Interface Solution for Remote Robotic Control and Monitoring Autonomous System

Hugo Perier¹^a, Eloise Matheson²^b and Mario Di Castro²^c

¹CERN, EN-IM-PLM, 1217 Geneva, Switzerland

²CERN, BE-CEM-MRO, 1217 Geneva, Switzerland


Keywords: Robotic, User Interface, User Experience, Web Design, Web Application, Software Solution.


Abstract: The area of robotic control and monitoring, or automated systems, covers a wide range of applications. The operating system, the kind of control, and the size of the screen used to present information to the user all vary in different robotic or industrial systems. This article proposes a system based on a user interface for real-time robotic control or monitoring of autonomous systems using web technologies laid on open-source projects. The purpose of this software is to be highly scalable over time and easily pluggable into different types of robotic solutions and projects. It must offer a high user experience and an appealing modern UI design, allowing technicians not expert in robot operation to perform interventions or maintenance tasks. The web environment provides an ideal platform to ensure the portability of the application so that it can be released on a multitude of devices, including laptops, smartphones, and tablets. This article introduces and describes the module, features, and advantages of the Neutron Framework. It presents how the users can interact with it and how to integrate this solution inside the CERN's Mechatronic Robotic and Operation solution.


1 INTRODUCTION

The increase of new technologies and innovation in the domain of digital technology has pushed robotics as a major research domain in 2022. While industry 4.0 is already a revolution (I-Scoop, 2022), there are many fields which will continue to be impacted by the innovation in robotics, such as education (Eguchi, 2014), logistics (Day, 2021), transport (Forrest & Konca, 2007), and the space industry (Katz, D. & Some, R., 2003), among others. One of the key points of a robotic solution is its ability to be controlled in real time by an operator or monitored in the case of an autonomous system. CERN's Mechatronics, Robotics, and Operation (BE-CEM-MRO) section is undertaking research and development on robotic systems for achieving interventions in hazardous areas. CERN's MRO section has developed its own framework for robotics, the CernTauro (Di Castro et al., 2018), also called the Cern Robotic Framework (CRF). The CRF is built as a micro-services

architecture that consists of a collection of separate components that operate on the system that powers the robot. It is possible to connect to and interact with the components using the TCP or UPD protocol. The CRF uses a structured way to exchange messages to receive information or to make requests to trigger action, with a bi-directional voluntary messaging system so that the client receives information about the status of their requests. The Neutron solution has been designed to cover a wide range of use cases and different scenarios, from planning to the realization of a robotic intervention. The software also allows the monitoring of an autonomous robotic system. It is possible for users to benefit from a rich API to create robotic behaviours using visual programming (Davis et al., 2011) to cover specific needs or for educational purposes. The program also contains a plugin mechanism that allows users to simply enhance the app's functionality in the event of unusual requirements.

^a <https://orcid.org/0000-0002-5201-9817>

^b <https://orcid.org/0000-0002-1294-2076>

^c <https://orcid.org/0000-0002-2513-967X>

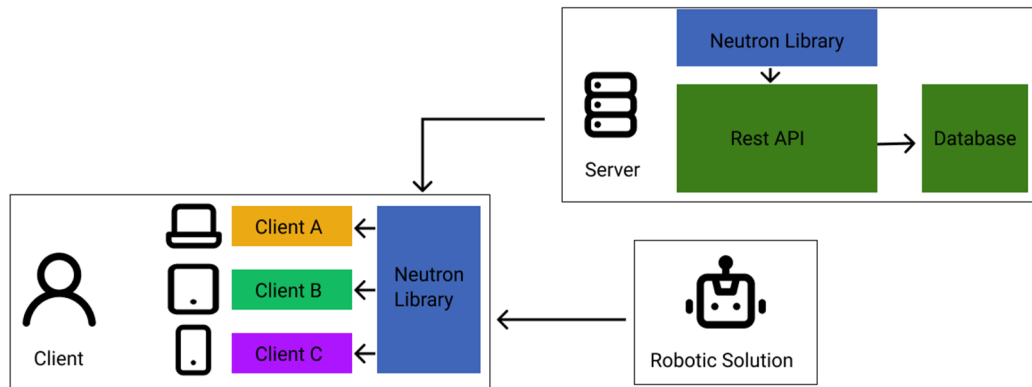


Figure 1: An overview of the proposed framework.

2 GENERAL OVERVIEW

The proposed solution includes multiple client applications that will be used by the end user, as well as a REST API (Li et al., 2016) running on a server. The setup of these clients consists of a combination of user-side applications (React and React Native) and custom backend code written in NodeJS to allow communication with the robotic solution, as well as our REST API. React and React Native are used to power the clients; the usage of these front-end frameworks allows for a high level of interoperability as well as software portability between computers, laptops, smartphones, and tablets. Electron is used here to embed the React web-application into a desktop application (Kredpattanakul, 2019). The Neutron Library, written in NodeJS and Typescript, provides each client with the tools they need to interface with the robotic system and access the REST API's services. NodeJS and Express have been used to create the REST API. This is designed to provide services to clients' applications, to centralize data, and to act as an inference server that can be used to run powerful machine learning algorithms. The desktop application may be used to organize a robotic intervention, including defining the aim and reasons for the intervention, the materials required, and the general prerequisites for a successful outcome. It is possible to define objectives and have an overview of the completion of those in the event of recurrent interventions, allowing participants to observe the overall progress unless the objectives are met. The database stores and centralizes all information regarding a previous intervention or the monitoring of an autonomous system. This enables powerful analytical capabilities and simplifies the process of debugging potential issues, as well as encourages iteration based on data acquired by robotic systems

(Cabi et al., 2019). The software's user interface and user experience are both built following the rules promoted by the atomic design principles (Frost, 2016).

3 SOFTWARE SOLUTION

3.1 HTTP Rest Api

The Rest Api is written in Node and is uses the Express framework. This API's primary objective is to provide services to the clients, such as an authentication system, information about the planification and progress of a robotic intervention, synchronization of configuration files, log aggregation, serving information about the connected robotic systems, or to be used as an inference server to benefits from powerful machine learning algorithms. The Rest API has security features to ensure that illegitimate foreign clients cannot connect to or transfer data to a robot. The information about the connection attempt is recorded every time a client makes a request to the API, allowing the administrator to keep track of the request flow. Protection against spam and SQL injection are in place to assure the API is not vulnerable from a malicious tier. To access the interface's functions, a client must be authenticated with the server. During the authentication procedure, an internet connection is necessary in order to get a token, which is subsequently provided to the robot so that it knows the client is trustworthy (Figure 2).

The process of gathering the logs of the robotic system in the database provides a powerful tool for data analysis. When the internet bandwidth permits it or when real-time control is off, actions performed on the software and in the robotic systems are registered

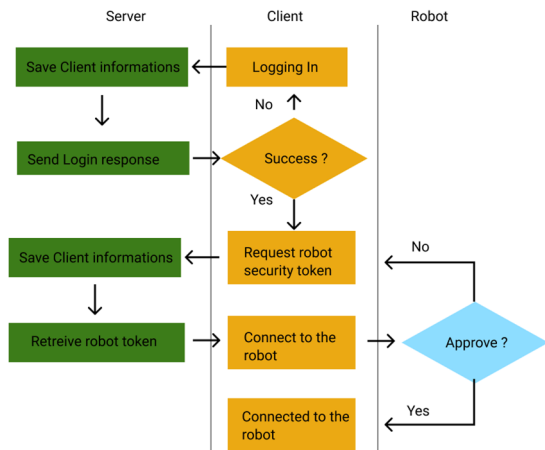


Figure 2: The connection lifecycle.

and forwarded to a server. These data can be analysed to help enhance the overall robotic system by identifying time-consuming human behaviours and recognizing recurrent patterns that might be automated (Cabi et al., 2019). This also provides a means of identifying and eliminating what went wrong in the event of a collision, a crash, or undesired robot or operator conduct. The database can also be used to track system metrics and as an operation history log that might be useful in future manual or autonomous operations (Fourie et al., 2017). The choice of MongoDB has been made because of its flexible schema requirements and its ability to handle large amounts of data efficiently by partitioning the database into shards, making it a very scalable solution as a database. It is possible to benefit from MongoDB Compass, which is an interactive and open-source tool for querying, optimizing, and analysing MongoDB data. This visual editing tool also enables users to understand and analyse data sets without a formal education in MongoDB query syntax and addresses MongoDB’s inability to natively support SQL. MongoDB’s data can easily be extracted into JSON format and then be analysed with external tools such as Python or R programming (Mahmood & Risch, 2021).

3.2 The Neutron Library

The Neutron Library provides all the necessary tools in order to connect, control or interact with the robotic solution. The library is written in Nodejs and Typescript. These technologies are easy to bundle, allowing the library to be used by all the available GUIs as well as the REST API. The main concept of the library is to create a Neutron Context. This context can be used to create one or multiple

connections to a robotic system. A connection is an object that contains a collection of clients, connected to the robotic system's processes. This object provides useful functionality for monitoring the connection's lifetime as well as exposing the robotic system's API. The library’s plugin-based approach allows to easily implement extensions during runtime.

3.2.1 Dynamic Protocol

The Neutron Library uses a strongly dynamic protocol in order to be able to support a large range of different robot operating system APIs. Adapters (Alves & Borba, 2002) are available to define the method of communication with one or multiple servers (Figure 3). This dynamic protocol can be defined as a combination of one or multiple transport layers (TCP, UDP, RPC) and a communication protocol (plain text, json, xml). The Neutron Library can handle multiple nodes, each using a different protocol. It is possible to connect to a REST server, a basic TCP server, and lastly, a ROS node, for example. Configuration files will be required to define the protocol of a robotic system, and an update will be necessary only when the protocol is modified. Configurations are stored and sourced on the Rest API and are synchronized across the many clients of the program used by the solution's customers.

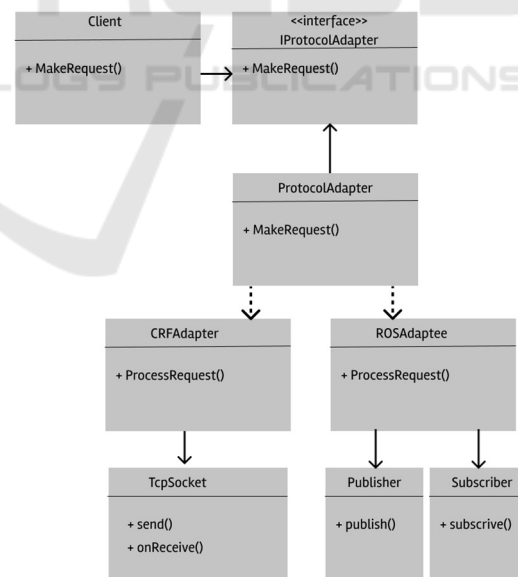


Figure 3: The communication and protocol adapters.

3.2.2 Capacitor

The Neutron Library provides the capability to operate as a capacitor. A client can make a request to the Neutron Library, which will be forwarded to a

specific host. A system like this is valuable for coordinating multiple robotic systems or dealing with the situation when one system is not connected to another. It is sometimes necessary to isolate a section of a system for security concerns. Using the client as a proxy allows to take advantage of such a system while maintaining its flexibility. By changing protocol while the request is forwarded, a web application hosted on a browser can communicate with a robotic system that does not support web socket-based communication protocol.

3.2.3 Plugin System

At any time, plugins can be mounted and unmounted. A class that implements the NeutronPlugin interface is referred to as a Neutron Plugin. The NeutronPlugin interface is designed to walk the plugin developer through all the requirements and options provided by the Neutron Library. A validation method is used to mount a plugin based on specified requirements or dependencies. Plugins are designed to enhance or modify the software's functionality. They can adapt and respond to an incoming message from the server or a request that is about to be made to a host. Plugins also allow the user to alter the behaviour of parts of the software's services, such as the robotic system's connection process, the logger, or, in some situations, to build custom functionality. Plugins can be loaded at the start of the software's execution or during runtime to add new functionality to the program. It is a clever technique to provide certain attachable and detachable features for a specific need (Chatley et al., 2003).

3.3 User Interfaces

The Neutron desktop application uses React as the front-end framework and Electron to turn the web application into a desktop application. React is an open-source framework created by Meta. React-based applications can be used as PWAs (Progressive Web Applications) (Bjørn-Hansen et al., 2017). Electron is used to transform the web application into a desktop application. It runs on top of chromium and the Node V8 motor. Electron is an open-source project and has been used for popular applications such as Microsoft Teams, Figma, or even VsCode. In order to use this user interface, the user must first be authenticated to the server. The user can then see information about future interventions and their general goals, as well as edit data about the ones they are a part of. Thanks to a novel modulable system based on components, it is possible to connect to a

robotic system from the interface and control it in real time. Fully automated systems can be monitored using the same modulable interface. It is possible to navigate through the interface and operate in real-time on a real-time system using a keyboard, mouse, or controller. The customization of the interface has been pushed so the user can assign key bindings to actions for more rapidity. Finally, a visual scripting framework has been implemented to enable the creation of robot behaviours using the Neutron API.

3.3.1 Authentication

The authentication view is the first page that the user sees. It is possible to both sign in and register from this page. A user who has just created an account must verify it via mail for it to become active. Because the administrator may have retrained the account creation to a domain, this verification ensures that the user is a member of the organization. During a connection, the software can be set to remember the credentials. During log in, the software will request that the server generates or refreshes the token that will be used to connect to a robotic solution and verify that the user is not a foreign client. A pop-up will appear if the user is unable to access the server because the token has expired.

3.3.2 Operation Overview

The Neutron GUI's home page is a dashboard that displays information to the user such as a calendar that notifies them of upcoming robotic interventions, charts that provide information about a task's general objective, and a list of possible connections to a robotic system via a direct connection or a customized one created from a configuration file. A mission editor is included in the interface for directly planning the prerequisites, objectives, and general procedure of a robotic intervention. Multiple users can be assigned to a mission. They will have access to its details and will receive the required information for its successful realization once it is completed. A mission is a recurring or one-time task that may be broken down into phases and completed by a system user under particular conditions. The manipulation and installation of equipment at a remote location, for example, is a task that would be manually confirmed by the individuals in charge of this stage. The data gathered during the robotic intervention might subsequently be utilized to confirm a second mission task autonomously. Missions may be set up to require specified materials, to be defined by a specific date or a range of dates, and to include attachments like pictures or PDFs. Mission information is saved on the

server and synced in real time among all users. It is possible to send a notification by mail to the parties involved when a modification is made to an existing mission.

3.3.3 Control and Supervision

From the dashboard, it is possible to open the robot connection menu. Users can connect using a standard configuration, a custom one, or by creating a connection from this view. A standard configuration handles the connection to a robotic system that is referenced by the server in the database. Those would generally be production ready and display information such as status, location, battery level, and information about the component mounted on them. Custom configurations are used in situations when the user wants to connect to a specific part of the robotic system using a different protocol or communication layer. This great level of customization for the connection to a robotic system is useful for development and testing purposes.

The real-time control and monitoring user interface is a set of adaptable, configurable, and interchangeable components. When a connection with a robotic system is made, the software will load the pre-defined components that will handle every functionality of the robotic system. Components can be used to visualize data received by the server or

stored in a database or execute actions in the robotic system in real-time. All the components rely on the same basis: a component can be opened, closed, reduced, dragged and resized according to some limit. A component can be configured to match the needs of its present application as closely as practicable. It is possible that different components cover the same functionality, but that does not imply that they will all be shown. Each component can be used in the way that the user wants. It is their responsibility to create the UI that fully satisfies the requirements. A configuration file can be used to save component selections, positions, and settings. This configuration can be loaded by a user to obtain the exact same interface setup. In order to improve the user experience, components also attach their actions to an input handler. The aspects and functionalities of a component will vary depending on its own size, the selected input handler, the mounted plugins, and the ability of the computer to access the Internet. Components are classified into two types: active and passive. When an active component is selected, key-bindings are activated, allowing the user to operate a real-time system with the best possible experience. Selecting an active robot arm control component, for example, would bind the direction of the joints to the active input handler's selected key. A passive component is mainly used to display information.

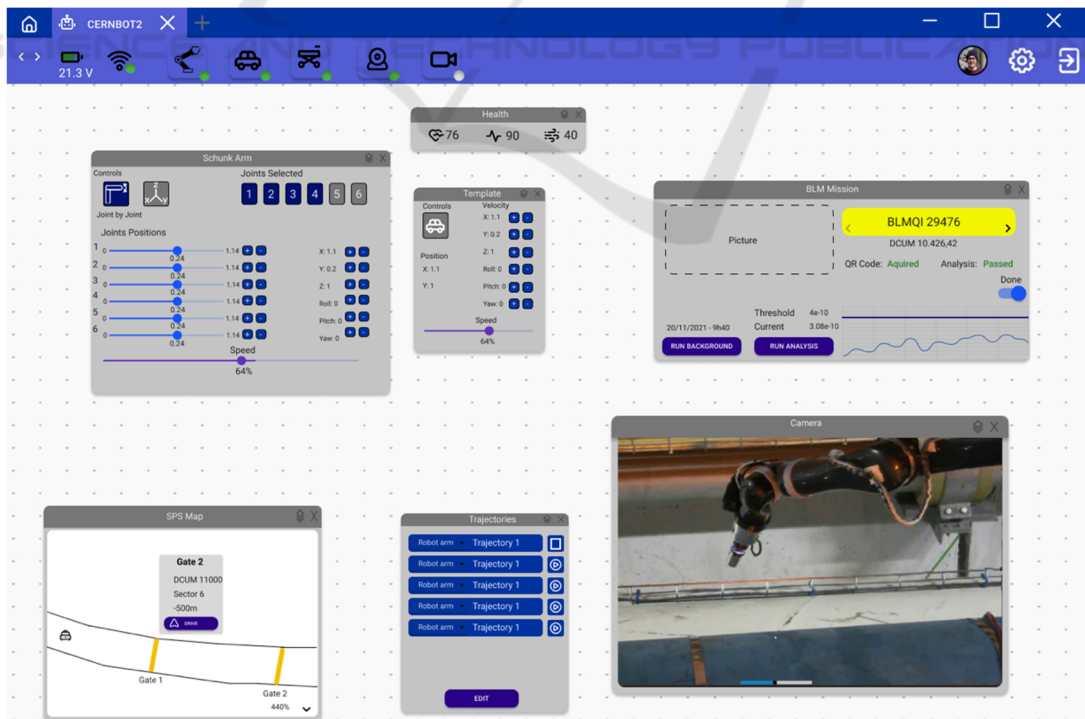


Figure 4: Multiple components are mounted in the real-time control view.

However, passive components can display buttons and inputs, but they will not bind keys to any actions. In Figure 4, the Schunk Arm component and the Robot Base component are active components. All the other components present in the figure are passive. Selecting the Schunk arm component will highlight it to indicate that it is active. The Schunk arm component will no longer be active if the user selects the Robot base component, and the key that was supposed to execute some action on this component will be disabled. The RobotBase is the only component that is highlighted and responds to inputs. The trajectory component is a passive component. It is possible to interact with this component to run a pre-recorded trajectory on one of the actuators available on the robotic system, but no binding will take place. While hitting keys on the input handler to conduct actions on the active component, the user may interact with a passive component.

3.3.4 Input Handlers

The Electron GUI supports a variety of input handlers. A keyboard with a mouse or trackpad, an Xbox or PS5 controller, or a handmade controller are all possibilities. The commander design pattern (Alexandrescu, 2001) is used in the software architecture that handles user input. When a user presses an input, the program looks for an action that is associated with that input.

In this case, this action will be produced, perhaps with parameters, and submitted to an action dispatcher, who will carry it out. The program is built to handle robotic operations with any input device that is supported. The user may simply switch between active and passive components without using the mouse since the UI components can change their look and bindings to the presently selected input device.

3.3.5 Visual Scripting

The Neutron Library includes a system of visual scripting (Figure 5). In the visual scripting view, users can arrange and configure blocks to construct basic or complicated programs that can subsequently be utilized as components in the control view. Visual scripting is a graphical way to manipulate objects and behaviour in the Neutron GUI without writing code from scratch. The logic is constructed by connecting visual nodes, allowing users or programmers to quickly develop intelligent and interactive systems. Visual scripting makes the process of getting a product from concept to production much faster and easier. Loops and conditions (if, while, and, or, nor, not...), math (additions, multiplications...), variables, actions, events, and custom nodes are the six main categories of basic visual nodes. The actions block refers to an actual action inside the software, such as sending a request to the robotic system or selecting a different input handler. A script can then be exported as a visual component or as a custom node that can be

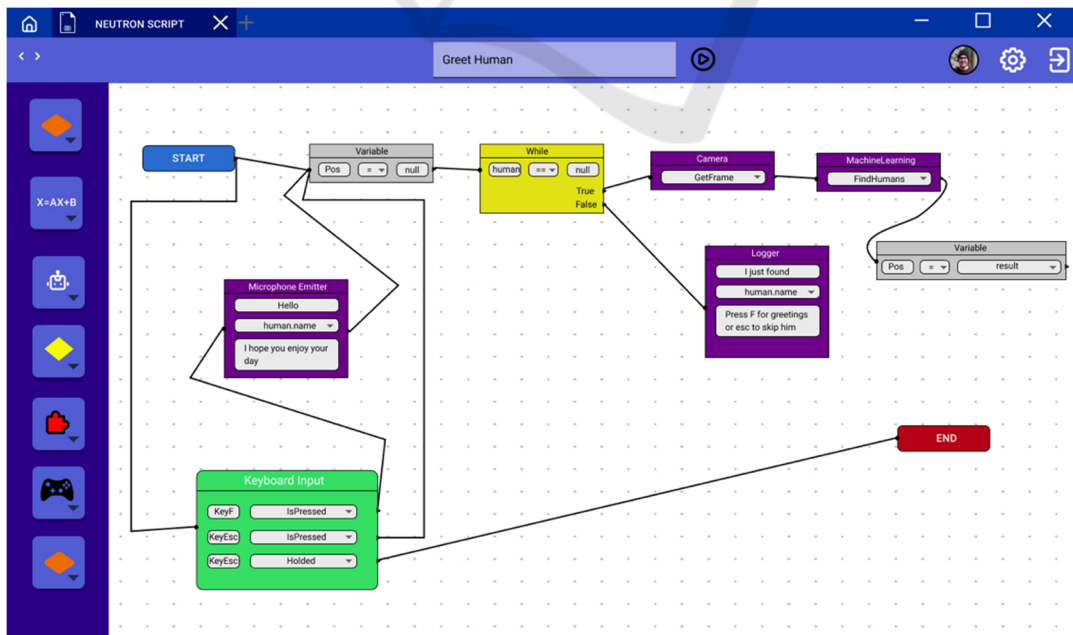


Figure 5: Implementation through visual scripting of the "Greet Humans" program.

reused in another script. When two nodes are connected directly, the second node receives the prior node's output value as input. Users do not need to assign and parse each output into a variable before reprocessing it into a second node since the system has been created that way. It is possible to assign the output, or a portion of the output, to a variable in some circumstances where this method is necessary. Every script must be composed of at least one start and one end node, defining the starting and ending points of the script.

The visual script presented in Figure 5 works in this way: The process begins on the "start" node. One way is through the "Keyboard Input" node. This node will register user input on a keyboard as events. When one of those events occurs, the following node is executed. A variable definition is the second block connected by start. The value of the virtual variable "human" is set to "null." It does imply that the variable has been registered; it exists but has no value. The flow immediately moves to a "While" node after defining this variable. The criteria "human == null" will be checked by this node. The nodes registered in the "true" output will be performed when this condition is true. Those nodes will request a camera frame, run a remote machine learning algorithm to recognize a person, and assign the algorithm's result to the "Human" variable. The "machine learning" node's output contains a JavaScript object, which is assigned to the variable "human." The while loop will cycle again if the "MachineLearning" node's output fails to retrieve a human in the provided image. Otherwise, the while loop will end, and the "logging" node will fire, notifying the user. Heading back to the Keyboard Input Node, if the key "F" is pressed on the keyboard after a human has been detected, the microphone mounted on the robotic system will speak the text described in the node, then the variable will be set to null again and the "while" node will restart, as requested by the software. The welcome will not be shown if the user pushes Esc. Finally, a lengthy press of the Esc key will bring the script to the "End" node, which will bring the entire visual scripting program to a stop.

3.3.6 Fault Tolerant System

The process of controlling a robotic system can be fraught with faults, timeouts, and delays. Some rules have been created to provide the user with an understanding of what is going on in order to promote consistency. When a button-triggered action is in progress, the button is disabled until the action is finished. Furthermore, when the user needs to be

aware of an issue or an essential message, a pop-up will display on the bottom right of the screen. In some cases, the pop-up will require the user's acknowledgement before it vanishes.

4 CONCLUSIONS

The proposed framework offers a scalable and highly distributable solution for the real-time control of robotic systems and for the monitoring of autonomous systems. Users merely need to extract the already used protocol, export it into the framework, and configure the appropriate settings to link this solution with an existing robotic solution. The use of the framework will assist administrators and operators from the planning stage to the realization of a real-time intervention. Operators can prepare for the intervention by customizing their user interface or by creating scripts and plugins, ensuring that the intervention is carried out under the best conditions. The framework also allows for autonomous system monitoring by customizing the dashboard so that all essential information is provided to the administrator, either on a screen installed on the machine or remotely on a computer. All the data collected by the program is centralized on a server to simplify the process of data-driven enhancement of the entire system and operator behaviour analysis.

REFERENCES

- I-Scoop (Ed.). (2022, April 4). Industry 4.0 and the fourth industrial revolution explained. I-SCOOP; www.i-scoop.eu. <https://www.i-scoop.eu/industry-4-0/>
- Eguchi, A. (2014, July). Robotics as a learning tool for educational transformation. In *Proceeding of 4th international workshop teaching robotics, teaching with robotics & 5th international conference robotics in education Padova (Italy)* (pp. 27-34).
- Day, M. (2021, September 12). In Amazon's Flagship Fulfillment Center, the Machines Run the Show. *Bloomberg*; www.bloomberg.com. <https://www.bloomberg.com/news/features/2021-09-21/inside-amazon-amzn-flagship-fulfillment-center-where-machines-run-the-show>
- Forrest, A., & Konca, M. (2007). *Autonomous cars and society*. Worcester Polytechnic Institute, 15, 23
- Katz, D., & Some, R. (2003). NASA advances robotic space exploration. *Computer*, 36(1), 52-61.
- Di Castro, M., Ferre, M., & Masi, A. (2018). CERNTAURO: A Modular Architecture for Robotic Inspection and Telemanipulation in Harsh and Semi-Structured Environments. *IEEE Access*, 6, 37506-37522.

- Davis, D., Burry, J., & Burry, M. (2011). Understanding Visual Scripts: Improving Collaboration through Modular Programming. *International Journal of Architectural Computing*, 9(4), 361–375. <https://doi.org/10.1260/1478-0771.9.4.361>
- Li, L., Chou, W., Zhou, W., & Luo, M. (2016). Design Patterns and Extensibility of REST API for Networking Applications. *IEEE Transactions on Network and Service Management*, 13, 154-167.
- Kredpattanakul, Y. (2019). Transforming JavaScript-Based Web Application to Cross-Platform Desktop with Electron. In *Information Science and Applications 2018* (pp. 571–579). Springer Singapore.
- Cabi, S., Colmenarejo, S. G., Novikov, A., Konyushkova, K., Reed, S., Jeong, R., ... & Wang, Z. (2019). A framework for data-driven robotics. *arXiv preprint arXiv:1909.12200*.
- Frost, B. (2016). *Atomic Design*. Brad Frost Web.
- Fourie, D., Claassens, S., Pillai, S., Mata, R., & Leonard, J. (2017). SLAMinDB: Centralized graph databases for mobile robotics. In *Book title is required!* (pp. 6331-6337).
- Mahmood, K., & Risch, T. (2021). Scalable Real-Time Analytics for IoT Applications. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)* (pp. 404-406).
- Alves, V., & Borba, P. (2002). *Distributed Adapters Pattern: A Design Pattern for Object-Oriented Distributed Applications*.
- Chatley, R., Eisenbach, S., & Magee, J. (2003). *Painless plugins*. London: Imperial College.
- Biørn-Hansen, A., Majchrzak, T. A., & Grønli, T. M. (2017, April). Progressive web apps for the unified development of mobile applications. In *International Conference on Web Information Systems and Technologies* (pp. 64-86). Springer, Cham.
- Alexandrescu, A. (2001). *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley.