



Article

Smart Platform for Monitoring and Control of Discrete Event System in Industry 4.0 Concept

Filip Žemla *, Ján Cigánek *, Danica Rosinová *, Erik Kučera  and Oto Haffner 

Faculty of Electrical Engineering and Information Technology, Slovak University of Technology in Bratislava, 84104 Bratislava, Slovakia; erik.kucera@stuba.sk (E.K.)

* Correspondence: filip.zemla@stuba.sk (F.Ž.); jan.ciganek@stuba.sk (J.C.); danica.rosinova@stuba.sk (D.R.)

Abstract: We are now living in a time when the fourth industrial revolution is bringing new technologies with intensive digitalization of all levels of production. This paper presents a complex solution for the monitoring, diagnosis, and control of the production process, in our case, discrete event systems. The proposed solution—a mechatronic platform—further develops and extends our recent results where the overall concept was outlined and some partial tasks were studied. The present paper provides the complete structure of the proposed platform together with implementation details for all functionalities. The developed platform uses the latest technologies, such as OPC UA; industrial communication and visualization protocols and tools; and augmented reality, which enables comfortable interaction with real processes. The laboratory-scaled case study shows the qualities of the proposed solution.

Keywords: mixed reality; PLC; Unity; manufacturing process; Industry 4.0; digitalization



Citation: Žemla, F.; Cigánek, J.; Rosinová, D.; Kučera, E.; Haffner, O. Smart Platform for Monitoring and Control of Discrete Event System in Industry 4.0 Concept. *Appl. Sci.* **2023**, *13*, 10697. <https://doi.org/10.3390/app131910697>

Academic Editor: Jose Machado

Received: 18 August 2023

Revised: 16 September 2023

Accepted: 21 September 2023

Published: 26 September 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The fourth industrial revolution, Industry 4.0, brings an integration of digital technologies to create more efficient, flexible, and interactive production processes. Typical features, such as digitalization, data-driven decision making, and transforming producers to smart factories, where monitoring and control is realized in real time, foster production efficiency, quality, flexibility, and responsiveness as well as resource optimization.

Constant progress in hardware technologies offers a higher performance in smaller dimensions, which widens possibilities for their use. The amount of data collected from production increases, which in turn requires the processing of large volumes of data. Therefore, a sufficiently powerful hardware system is needed, which favors the use of cloud technologies. If all requirements are met, a digital production environment can be developed [1,2].

Augmented and mixed reality are recently developed technologies that provide functions and services that are extremely useful for industrial applications. This technology also forms a significant part of our project and its inclusion in the proposed platform is one of the main contributions of this work. Therefore, this technology, together with communication between individual smart devices, is at the center of the introductory analysis of the proposed solution. Since our work covers several technologies and fields, the analysis can be divided into two parts [2–4]:

- Analysis of communication between smart devices;
- Mixed and augmented reality (AR).

1.1. Communication between Devices

In this area, we have studied possibilities and technologies enabling the communication of smart devices through the local server to the cloud server. The aim of this analysis is

to choose appropriate communication protocols as well as cloud technologies. An extensive survey of software-defined networking and related technologies and services can be found in [5].

Several recent results concerning this topic have been studied. The factory simulation [6] represents communication from the production level equipped with a PLC to the local server (KEPServerEX) and then to the cloud server (ThingWorx). Thus, the augmented reality application is fed with data from the cloud. This solution uses the Ethernet and OPC UA protocols for communication and two servers (a local KEPServerEX and a Thingworx cloud server). The control computer with the SCADA system receives data from the PLC device via the OPC UA protocol, which are then sent to the KEPServerEX and from this local server finally to the cloud server Thingworx via an Ethernet protocol and back to the AR device on the local network.

Another experimental system was an energy trading platform with peer-to-peer communication [7]. This project is based on remote access and the remote control of the devices. The implementation uses the MQTT protocol for peer-to-peer communication, and two local servers are employed (an MQTT broker using Node-RED and a web user). Communication is initiated by the user's login.

1.2. Mixed and Augmented Reality

Two main issues concerning augmented reality have been studied: how to adequately present the objects in augmented reality and how to affix virtual objects within the physical world. Project [8] is dedicated to the presentation of new construction parts via AR. The proposed solution offers the possibility to include various geometric shapes and 3D objects and place them at any position in space. The size of the added objects can be varied. After finishing the design, the model can be saved. The saving function scans the whole object and records the video of the scanned object from different angles. This video in the form of an image series is sent to the cloud server. The saved video then serves for a recognition of the respective machine (device).

Another project studies a robot control using augmented reality [9]. In this case, the project includes several robots that have Raspberry Pi modules and can be controlled using an AR application by drawing a driving path. The application consists of a preloaded scene that can be placed on any flat surface. After the scene is loaded, individual robots can be controlled by drawing a path on the tablet. Since simpler Raspberry Pi modules are used, communication takes place via the TCP/IP protocol.

In [10], a mobile device is presented that is equipped with augmented reality to present measured values delivered by sensors. These measured data serve to detect malfunctions and alarms. The recorded data can be shared by other devices using augmented reality or employing a web browser.

Also worth mentioning is the patent [11] from Meta, where it is stated that an AR application for an AR headset accesses a library of user interfaces for groups of devices. After recognizing an object in space, it simply loads the interface for that object.

Another inspirational work is [12], where AR is combined with the IoT and cloud technologies. The MQTT protocol is used for communication, and the Unity RD engine is used as a development tool. One of the main issues—object recognition—is solved using Wikitude Studio to create a 3D map. The AR application communicates with a cloud server and a digital image of the studied object is saved.

In 2022, the IIoT Situational Data Visualization project was developed into the MS HoloLens 2 device [13]. The project presents data collection from the PLC device of an industrial furnace, which is then sent to a local server. The authors use the OPC UA protocol, and the local server based on the Node-RED tool provides visualization created in the Dashboard library. The local server then sends data to the MS HoloLens 2 device using the MQTT protocol. The data are subsequently displayed on the production system with the option of displaying the data in the Dashboard via a web browser.

Last year, a project [14] was published that deals with modern methods for human–machine interaction. Several options are listed in this publication, such as the use of artificial intelligence, mobile devices, the IIoT, Industry 4.0, and, especially, the possibility of scanning real objects and subsequently transforming them into 3D objects.

Our recent works [15,16] were devoted to partial tasks concerning the digitalization of the production line within the Industry 4.0 framework. In [16], a data acquisition and visualization SCADA system is proposed for a laboratory production line. The measured data are transferred and collected in a cloud database based on the Microsoft Azure platform. The basic concept of a digitalization platform for event systems was outlined in [15], where basic notions and some proposed technologies were studied; however, there was no concrete solution or implementation.

The main goal of this paper is the design and implementation of a complex smart mechatronic platform, where monitoring and data recording is realized using cloud services. The proposed solution further develops our recent results, extends them, and includes augmented reality as a means for communication and interaction between the production line and the monitoring and control system. A recognition of the individual production line is also part of the proposed solution. The monitoring and analysis of measured values can be followed by the web application as an inherent part of the developed platform. The developed platform functionalities were tested on laboratory production lines equipped with industrial PLCs.

This paper is structured as follows. In the preliminaries section, we focus on the history of industrial production up to current trends in Industry 4.0 to motivate our research. Basic notions and requirements are introduced as well. Section 3 presents this paper’s main contribution, the detailed design of a complex smart mechatronic platform and its implementation. The obtained results are summarized in Section 4.

2. Preliminaries and Problem Formulation

We are living at the time of Industry 4.0, which brings a lot of changes, such as smart devices in production processes. With the use of cloud databases and communication with individual smart devices, we can record any activity of machines and devices in industries and provide application users with all the necessary data.

The term Industry 4.0 was first used in 2011 at the Hannover Messe. The main idea was to use the Internet in all aspects of production. In 2014, the concepts and progress in achieving the vision of the new industrial revolution, which Germany was already leading at that time, were presented. Since 2014, industries in Germany have been using the term Industry 4.0. More than 40% of companies worked with this concept through concrete initiatives. Such cooperation was mainly helped by 28 sponsors and partners, such as Siemens, Cisco, Phoenix, Beckhoff Automation, and others. At the conference, the need for standards was also addressed, such as mechanical, pneumatic, physical, or communication standards. Among the cited standards, concepts such as OPC UA, WSDL, EDDL, and IEE 61499 were mentioned [15–20].

Augmented reality belongs to the innovative technologies of Industry 4.0. However, notions of virtual, augmented, and mixed reality are very often confused or mislabeled. First, let us explain the differences between the individual terms. People perceive two types of reality, physical reality and computer-generated reality, which is also referred to as XR (eXtended Reality). XR is a universal term covering all virtual reality connected technologies. Extended reality includes all three other types: virtual (VR), augmented (AR), and mixed (MR) realities [21–23] (see Figure 1).

2.1. Virtual Reality (VR)

Virtual reality transports the user from the real world to a virtual world that simulates visual and sound sensations. In virtual reality, the user of the application ceases to perceive the physical environment in which they are physically located. The virtual environment is simulated and can often be distant in time and space from the physical environment; it is

not connected to this environment. Today, virtual reality is often used in the game industry, along with teaching and education, for the simulation of various situations [21–23].

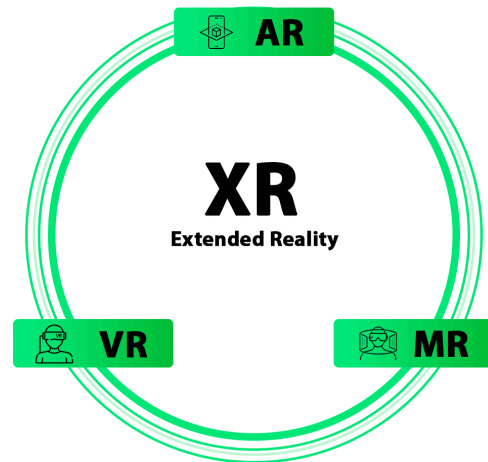


Figure 1. Extended reality technologies.

2.2. Augmented Reality (AR)

Augmented reality, unlike virtual reality, is a computer-generated reality embedded in a physical environment. As the name suggests, the physical environment is supplemented with virtual elements that we can interact with. Augmented reality is generated using mobile devices or headsets [21–23].

2.3. Mixed Reality (MR)

Mixed reality, often referred to as hybrid reality, is similar to augmented reality and is set in a physical space with virtual elements. However, the interaction of virtual elements can also affect content from the real world [21–23].

2.4. Industry 5.0

Currently, we are encountering the new concept of Industry 5.0, which places people in the foreground within the work process, the so-called “Human-centric solution”. The case study from this paper corresponds to Industry 4.0 and a person intervenes in the management through the Microsoft HoloLens v2 device.

Industry 5.0 differs from previous concepts by focusing on the collaboration between humans and machines in the production of products and services. In this new industrial era, technology is becoming more personalized, interactive, and adaptable, allowing people to work with it in real time and contribute to innovation and process improvement.

In Industry 5.0, we expect to see more collaborative robots and automated systems that can work with humans and support them in production and service delivery. This should increase the production efficiency and quality, reduce costs, and improve working conditions [24–26].

Based on present technologies and solutions, especially those related to augmented reality in the IIoT and clouds, the main aim and requirements for the proposed smart platform have been determined as follows [25]:

- Device communication will be implemented only in accordance with IIoT standards;
- The user environment will be displayed dynamically based on the detected device;
- Data for remote browsing will be provided using an Internet browser.

From the acquired knowledge, we were able to further determine the requirements for our system, which we divided into functional and non-functional ones:

Functional Requirements:

- Communication between a device with mixed reality and a local server using the MQTT protocol;
- The possibility of communication between the local server and the production model based on the OPC UA protocol;
- The development of an application for mixed reality in the Unity environment;
- The use of the MS HoloLens 2 headset device to display mixed reality;
- The dynamic display of the visualization of individual production models based on definition schemes;
- The recognition of the production model based on its shape and dimensions;
- The intuitive visualization and management of all production model data in mixed reality;
- The provision of data for remote visualization using a web application;
- The provision of data visualization and the control of production models using touch screens.

Non-functional requirements:

- Definition schemas stored in the JSON format;
- The use of the Azure cloud platform for server management.

3. Design (Hardware and Software) and Implementation of Mechatronic Platform

The development and design of the platform is based on the study of available technologies, analysis of the current state, and existing solutions. The platform uses a wide range of technologies and therefore it is necessary to divide its design into several units. In this section, the development and design details are presented. Since all parts of the platform interact with other parts, we recently proposed the project structure of this platform, which is shown in Figure 2 [15].

In Figure 2, the color-coded parts are depicted according to the tools used. The first of them is the automation of production lines and the creation of an HMI in the TIA Portal [27] environment (an orange area). The second is the creation of a local server for processing data from PLC devices and the subsequent provision of these data to other parts via another communication interface (a blue area). The next part is the creation of a cloud server on which analytical data are stored, which is further provided to the client's remote device (a turquoise area). Another part is a frontend application created in the Angular environment for the remote visualization of analytical data (a black area). The last part is the Unity [28] application for the MS HoloLens 2 device (a yellow area).

The project contains several types of physical devices on which the applications from the previous section of the text are located. Communication is therefore designed, including application communication as well as mutual communication between individual devices. Figure 3 shows the communication, the direction of sending information, and the interactions of individual devices.

The proposed method of data transmission and processing is shown in Figure 4. In the proposed concept, a mixed reality display device recognizes real objects using a camera. After recognizing them, it connects to the mechatronic device and requests initialization data. The local server serves as an intermediary between the mechatronic system and the mixed reality display device, communicating with both devices using different communication protocols. It also sends analytical data to a cloud database, which provides stored data for remote data visualization in a web browser.

Individual parts of the project structure and their applications, physical devices, and creation will be presented in detail in the next section.

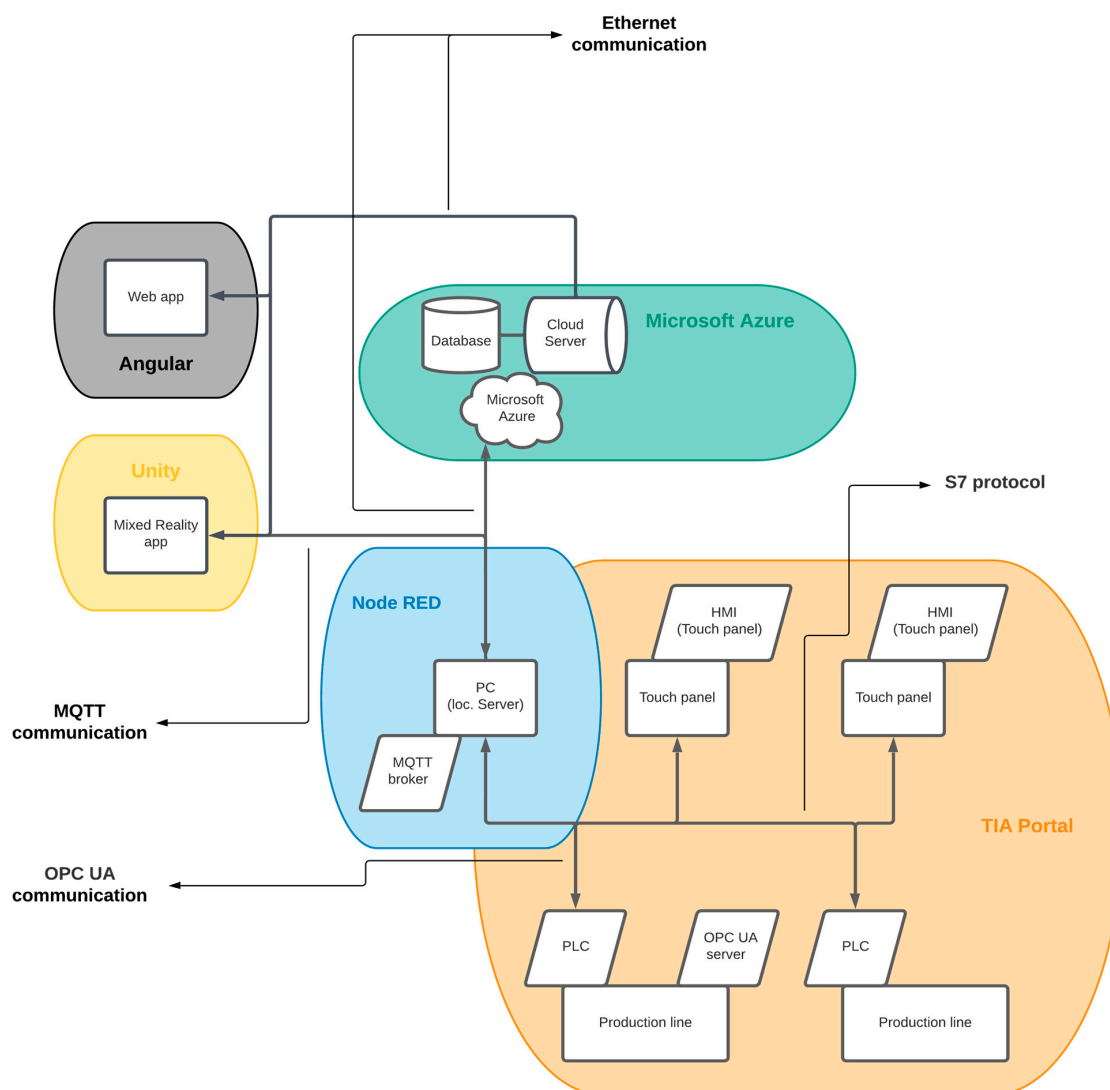


Figure 2. Project structure (according to [15]).

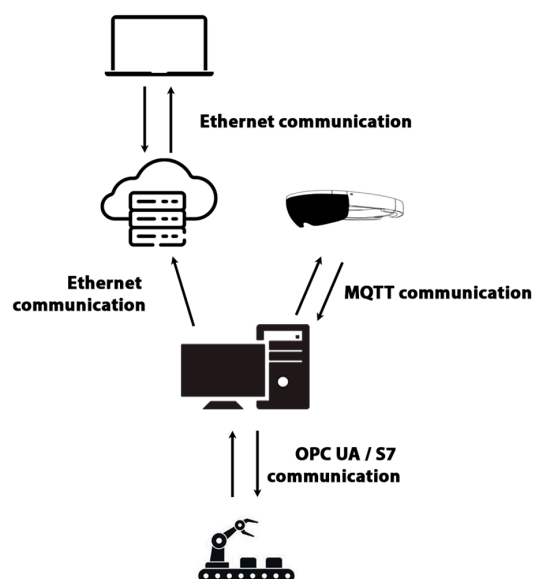


Figure 3. Device communication.

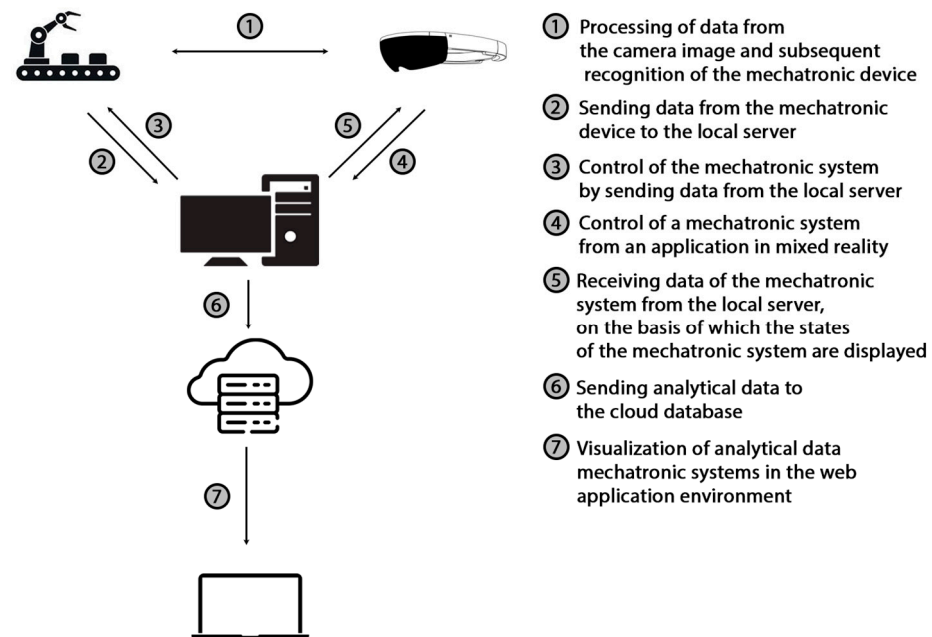


Figure 4. Control and visualization of mechatronic systems.

3.1. Automation of Production Processes

An important part of the project was the appropriate selection of laboratory production models and PLC devices. We chose PLC devices from Siemens. They provide communication using their own S7 protocol on the devices and communication using the OPC UA protocol on higher models (S7-1500 and some S7-1200) [29].

When designing the platform (system), we proceeded in accordance with the principles of Industry 4.0. In this case, we require the proposed system to be modular. Therefore, we chose communication using the OPC UA protocol, which is also compatible with devices from other manufacturers. Two laboratory models of production systems are used for testing, so we decided to use both protocols to verify modularity: one model will use the S7 protocol and the other will use the OPC UA protocol.

As already mentioned, both laboratory models are automated using Siemens PLC equipment and each of them contains a touch screen Siemens HMI KTP 700 Basic [30] for controlling the model and displaying its data.

All devices communicate at the level of a local network without access to the Internet. The devices can communicate with each other, but, since they are two separate systems, the communication will take place between the PLC device, the HMI device of the given laboratory model, and the local server.

In the proposed solution, we use educational models from Fischertechnik [21,31] as production models. Individual laboratory production models, their visualization, and their functionality are described in detail in the next section.

3.1.1. Production Line Model

The first laboratory production model is the U-shaped production line model (hereafter referred to as model 1). The production line model represents the process where products are processed by two machines. The model consists of four belts, two machine tools, and two sliders, with both sliders allowing forward and backward movement. Five light sensors are available to detect the position of the product in the production process. To detect the positions of the sliders, the model is equipped with four end sensors (two rear and two front). All the listed components of the laboratory production model are marked in Figure 5.

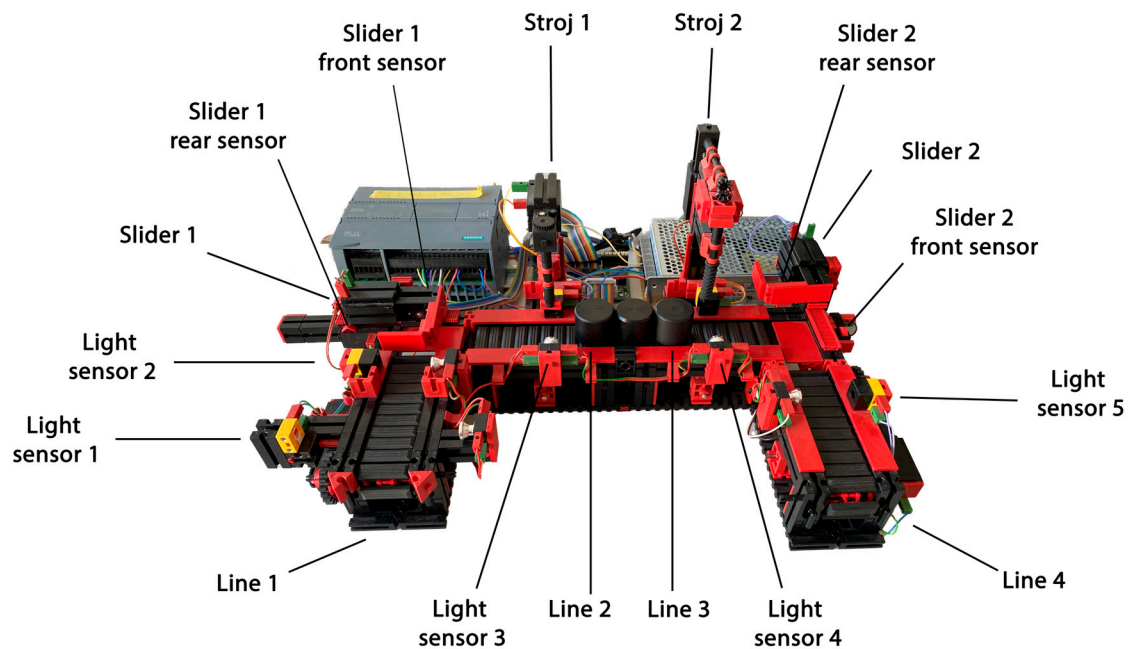


Figure 5. Production line model (model 1).

A smaller PLC device, model S7 1215C DC/DC/DC, was used to control this model. A PLC device of this type contains 14 digital inputs, 10 digital outputs, and 2 analog outputs. For the automation of this system, 9 digital inputs and 10 digital outputs were needed.

For this laboratory production model, we compiled an algorithm in the TIA Portal environment, where we designed the processing of the product and its movement through the entire production process. Furthermore, the algorithm contained the following treatments and functionalities:

- The treatment of the collision of products on the production line with several products simultaneously located on the production model;
- The detection of product loss from the belts: if the next step of the control algorithm is not performed within 5 s, the alarm on the given belt is activated;
- Counting engine hours for all belts and machines;
- Counting the number of products on the production line and the number of finished products;
- The manual resetting of both product numbers;
- The possibility of operating the production line in manual and automatic modes;
- The start function, a treatment for a quick shutdown of the line in the event of an unexpected collision.

3.1.2. Sorting Line Model with Color Detection

Another laboratory production model is the color detection sorting line model (hereafter referred to as model 2). This system provides product storage based on color recognition. It consists of active members, such as a belt, a compressor, and three pistons. Product color recognition is performed by a color sensor, and information about the product's position is provided by five light sensors. All listed parts of the laboratory model are shown in Figure 6.

For model 2, we used a powerful PLC device, model S7-1516-3 PN/DP, which provides a higher performance and many more functions than the PLC in the previous case for model 1. The S7-15xx PLC devices themselves do not have inputs and outputs. Therefore, it was necessary to assign input–output modules to the PLC device. In Figure 6, we can see the control system that we designed and implemented, where the PLC device is connected in combination with four modules and one industrial source. The input–output modules

provide us with 32 digital inputs, 32 digital outputs, 8 analog inputs, and 4 analog outputs. The used PLC and output and input modules are placed separately and connected to the controlled production line using wirings.

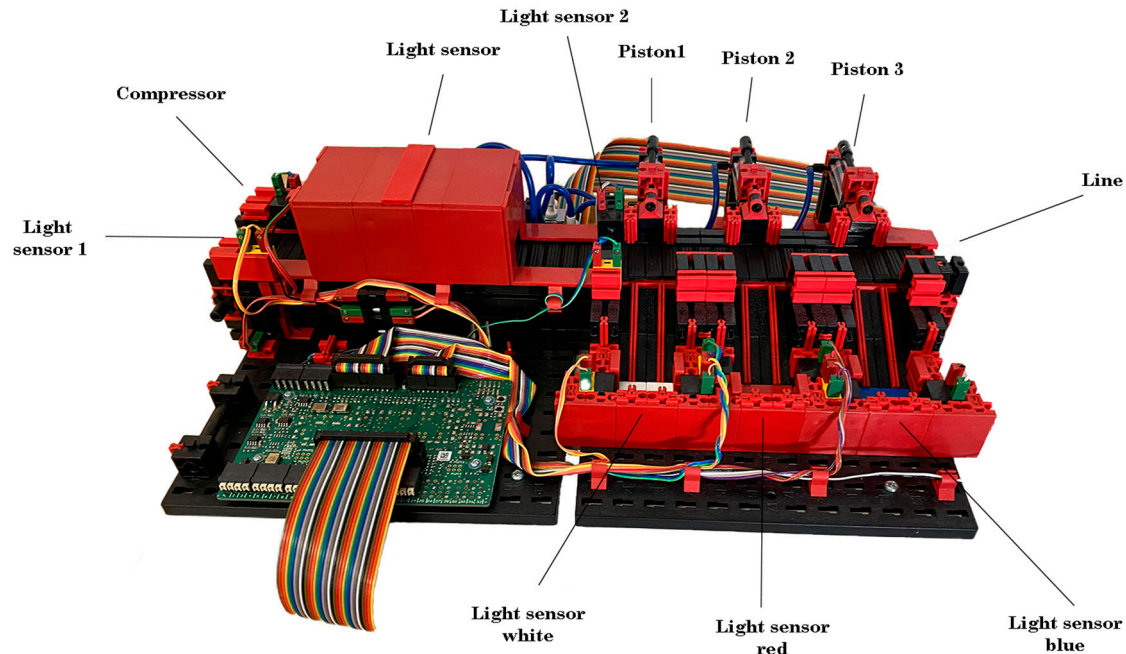


Figure 6. Sorting line model with color detection (model 2).

The last step was the compilation of the algorithm and visualization for model 2. The model is designed for three types of products, red, white, and blue ones. After placing the product on the line, the product moves to the position of the color sensor, where the color of the product is read within 0.5 s.

The algorithm also includes the following functionalities and treatments:

- Collision detection for the full storage space—in the case of a full state, it triggers an alarm;
- A start function—a treatment for a quick shutdown of the line in the event of an unexpected collision;
- Multiple control modes—manual, automatic, and semi-automatic modes;
- Counting engine hours for all action members;
- Counting sorted products according to the recognized color and the total number of products.

3.2. Local Server Development

In our case, the local server needs to process data from all PLC devices in the system, store them in the database, and provide data to portable devices. Therefore, we placed great emphasis on the possibility of communication using several protocols and on the necessary performance for data processing.

For our communication requirements (Figure 3), it was ideal to use the Node-RED tool for creating a local system. Node-RED [32] is based on a combination of Javascript and modular programming (connecting logical cells). Node-RED itself does not contain all the necessary tools, be it for communication, creating graphics, or various logical functions. Therefore, the implementation of third-party libraries was necessary. Below, we list the libraries we used:

- node-red-contrib-opcua—a library for creating an OPC UA server and enabling simple communication on this protocol;

- plcindustry—a library for enabling communication with Siemens devices based on the S7 protocol;
- node-red-contrib-mssql—a library for communication with MS SQL databases;
- node-red-contrib-aedes—a library for creating an MQTT broker and communication based on the MQTT protocol.

From the libraries used, it follows that the local server performed four functions, specifically, four different communications based on different protocols. It could be said that in our case the local server is the heart of the project, which connects all units together (Figure 2).

3.3. Cloud Server Development

The next part was to develop and configure the cloud server. In our case, three functional requirements were placed on the cloud server:

- Enabling data storage;
- The provision of data to applications;
- Providing a web application for the public.

We had a choice of several cloud platforms, but, according to the analysis, we decided to use the MS Azure platform [32]. MS Azure provides many services, from which we have selected the following:

- SQL server + SQL database;
- App Service;
- Static Web App.

Below, we describe in detail all the services, their design, their configuration, the tools used, and the applications that we have uploaded to the cloud server.

3.3.1. SQL Server + SQL Database

First, it was necessary to create a database in which the data that are provided for remote browsing are stored. From now on, we call these data analytical data. So, we developed an SQL server, called *plscwithlove*. We stored this server in a data center and together with the server we also created a 2 GB MySQL database. Next, we set up the SQL server as a private server, to which we added the IP addresses of the computer on which the local server was running and the IP address of the server with the backend application. Lastly, we set up security rules for authentication using credentials. We chose the login data according to the general instructions. The password was composed of more than 15 characters and contained numbers, lowercase characters, uppercase characters, and symbols.

Once the server was configured, we could create the database structure. Each of the production lines contained three groups of analytical data: they were the engine hours, number of products, and alarms. Accordingly, we adapted the structure of the database (Figure 7). We created six tables in which each row corresponded to one entry. Each of the tables contained a primary key index that was filled in automatically. The other values corresponded to the types of variables that we wrote in the table.

3.3.2. App Service

The next development step was to ensure access to the data using the backend application. Uploading the application was made possible by MS Azure [33] through the App Service. Here, it was necessary to configure the server environment. We chose the operating system under which the server should work, the Java version of the platform, its location, and which project should be uploaded to it. We chose the location in the data center. Linux, which is compatible with Java applications, was chosen as the operating system, and the Java version of the Java 17 SE platform. As the source of the project, we chose a repository on the Github platform [34], where we later uploaded the project. In this

case, the project is uploaded automatically after the new version of the project is uploaded to the repository.

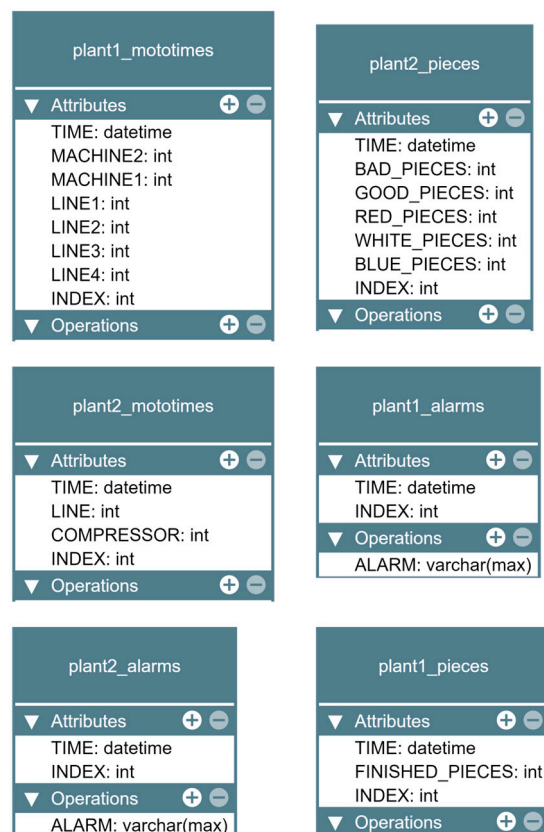


Figure 7. UML diagram of a cloud database.

The backend application had to be created and tested in the initial phase on a local environment. We used the IntelliJ tool for development, used the same version of the Java platform as on the server, and created a new project in Springboot framework version 3.0.4 [35]. First, we connected the application to the MySQL cloud database by implementing the mssql-jdbc library and entering the login credentials. We used the springboot-hibernate library to work with data in the database. With this, we created the basis for our project, and we could further work on creating endpoints for working with data.

The project was based on the REST API architecture in the Springboot framework. For each of the tables, it was necessary to create a model with all attributes. Using the lombok library, we could easily add getter and setter methods for all models using annotations.

Next, we created a repository that directly queried the database. Just like the models, the repositories had to be created separately for each model. Since the repository in the Springboot framework serves as an extension class of the JpaRepository class, there was no need to add new features. All necessary functions were already included in the JpaRepository class. Furthermore, we created services for individual production lines, which provided us with data obtained from repositories for controller classes. In general, the architecture is designed so that the controller configures the path and type that the application queries. Data processing will take place in the service and the repository will provide the data of the given model. In the controller, as already mentioned, we set the endpoint mapping and the http method it responds to. Since data were provided from the backend application, it was a GET-type http method.

We then tested the backend application using the Postman tool, where, after calling the endpoint, the backend application returned data from the database.

3.3.3. Static Web App

As in the previous service, this service also required setting up a server and setting the source from which the frontend application was uploaded. The procedure was the same as in the previous case: we created a new repository on the GitHub platform and connected it to the server. After creating the server, we could start developing the frontend application.

We decided to make the frontend application in the Angular JS framework [36], version 15.2.3. Currently, a very powerful tool for appearance development is the NgPrime library, which we used in our project to define the appearance of elements on the website.

Before developing the application, it was necessary to build a use case diagram, according to which the application was built. Designed diagram includes two types of users: logged-in and unlogged users. Since the web application is publicly available at the web address <https://wonderful-moss-094cedb03.2.azurestaticapps.net/>, e.g., accessed on 6 August 2023, it was also necessary to consider unauthorized access. Therefore, a non-logged-in user can only access the main page and login. In addition to logging in, the application will also include functions such as light/dark mode switching, the display of all lines, and the display of the analytical data of lines in a table and in a graphical interface.

The application was designed to meet current web application standards; be sufficiently clear, intuitive, and minimalist; and avoid unnecessary loading. The designed application can be adapted to mobile devices of different sizes as well as personal computer screens. The application was built using the Angular JS framework, so when the content changes, instead of loading the entire application from the server, only the content that changes is dynamically updated. In the application, we used the angular-router library, which uses the concept of routes to control navigation within the application. Routes are defined as a combination of a path (usually a URL segment) and a component that should be displayed when navigating to the path. We also used the lazy loading technique, which allows Angular modules to be loaded as needed, instead of loading the entire content when the application is launched. This can help improve the initial load time and overall performance of an application, especially in cases where it has many components, services, or other resources. In our application, the initial app-component module is loaded, which contains the main page component and the login component. After the user logs in, the private module with the components of the private part is loaded (see Figure 8).

As already mentioned, the application can be divided into three groups. They are the basic part of the application, which is found throughout the application, the public part, and the private part. Accordingly, it is possible to divide the components into a diagram. In Figure 8, we can see a diagram of the components, where the individual parts are distinguished by color. The basic part (yellow color) consists of a header, color mode switch component, navigation (breadcrumbs), and routing. The public part (green color) is accessible to a non-logged-in user, and it includes components such as the home page and login. We present the individual subpages in detail below. The private part (blue color) is accessible only after the user has logged in and includes data visualizations of laboratory production models.

After the initial loading of the web application, we get to the initial page (Figure 9). There is a header where we can switch the state of the color mode or log in. After clicking on the login button, the application redirects us to the login form. Since the developed platform is based on the Industry 4.0 concept, there is also an image with accompanying text about Industry 4.0 on the opening page.

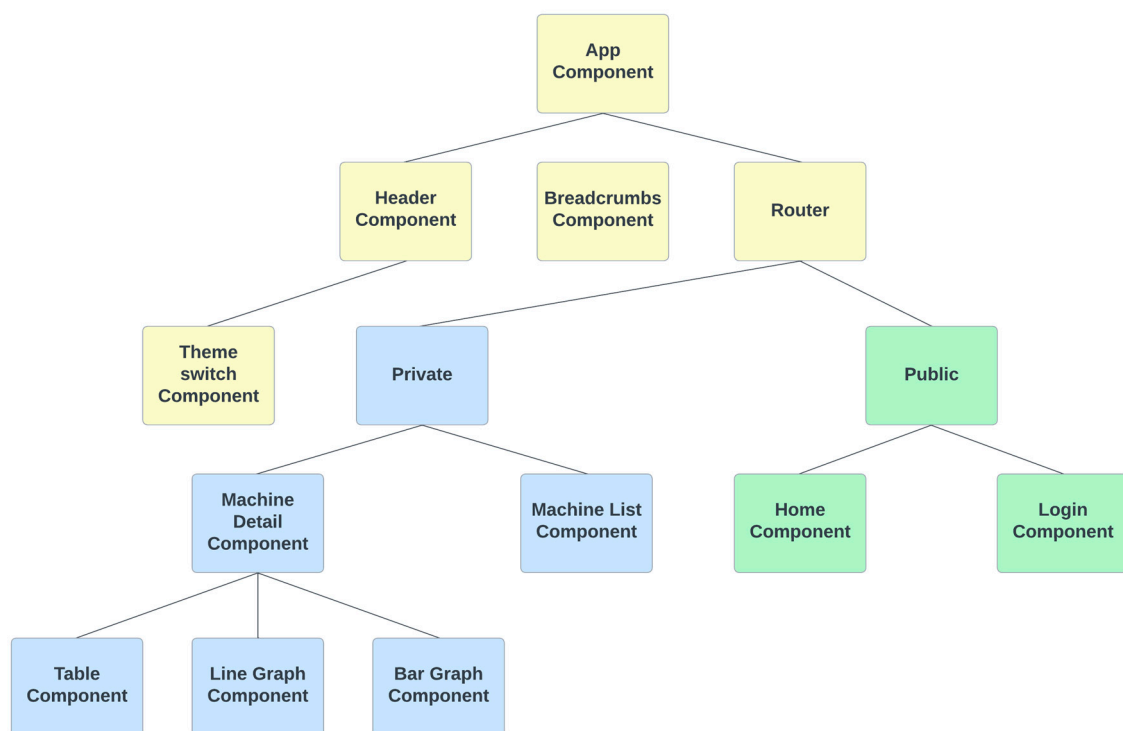


Figure 8. Diagram of components in the frontend application.

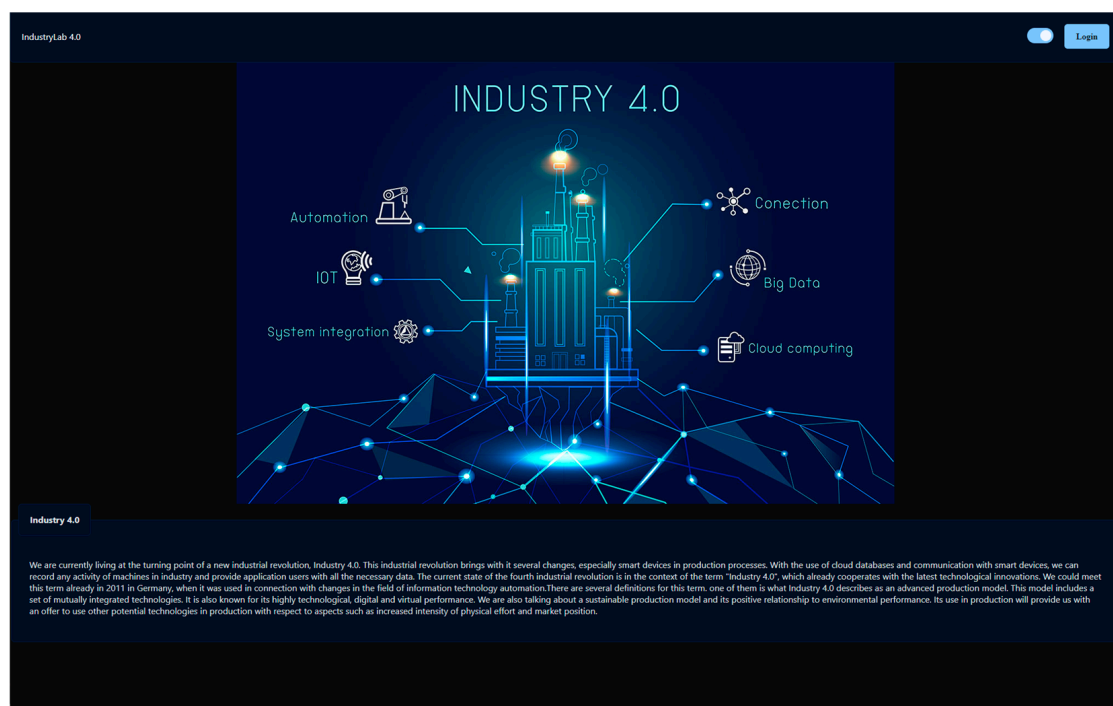


Figure 9. Home page of the frontend application.

After logging in, the user gets to the list of production lines, which is shown in Figure 10. This subpage offers the user a quick overview of all the production lines in the system. Each of the lines provides an image of the production line, its name, and a brief description. It also contains buttons that redirect the user to the display of the individual data that the given production line contains.

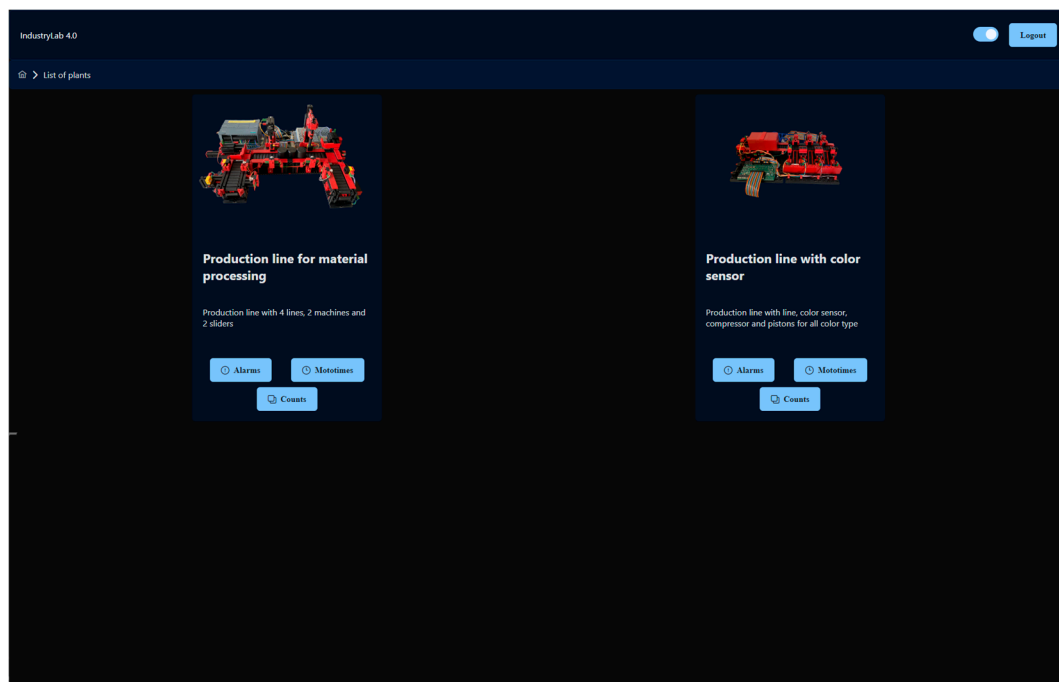


Figure 10. List of production lines.

The display of the production line data is represented in tables (Figure 11). Individual columns can be sorted from smallest to largest (numerically, chronologically, or alphabetically). It is also possible to sort the data according to the entered value. The tables also allow pagination and the option of choosing the number of displayed rows. For each variable from the production line, it is possible to change the data type in the submenu. Each type of data also allows a graphical display of the data after configuration, which is described below. If the data provide a graphical display, a button with the corresponding icon is displayed above the table. The graphical presentation of the data is possible in two ways, a line graph (Figure 12) and a pie chart (Figure 13). The line graph allows the display of data in a time plane with the possibility of zooming. The pie chart displays specific data in one record and allows the selection of the given record using the select html element. Every change of data in the graphs is performed with the help of animation.

Line 1	Line 2	Line 3	Line 4	Machine 1	Machine 2	Time
0	0	Match All	0	0	0	18:10:00 21.02.2012
0	0	Equals	0	0	0	23:09:17 29.12.2022
0	0	Equals	0	0	0	23:18:01 29.12.2022
14	0	Greater then	0	0	0	00:00:00 30.12.2022
14	0	Less then	0	0	0	01:00:00 30.12.2022
16	0	Clear	0	0	0	01:20:23 30.12.2022
16	0	Apply	0	0	0	02:00:00 30.12.2022
16	0		0	0	0	03:00:00 30.12.2022
16	0		0	0	0	04:00:00 30.12.2022
0	0		0	0	0	14:00:00 30.12.2022

Figure 11. Table with data.

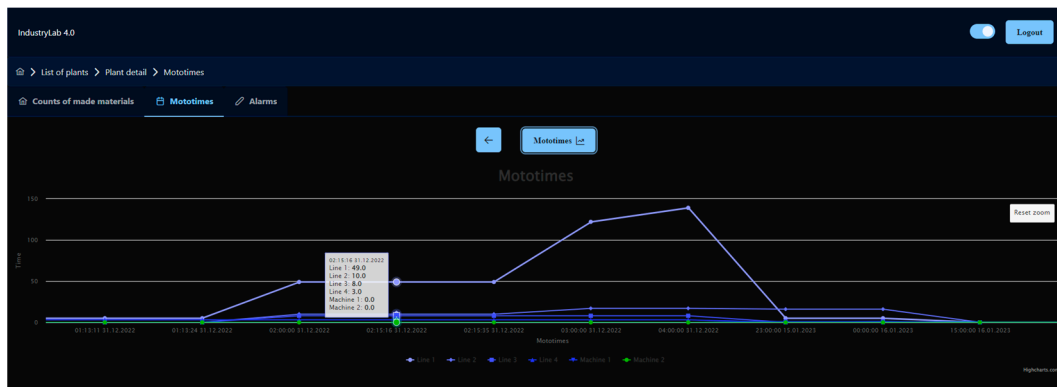


Figure 12. Line graph of frontend application.

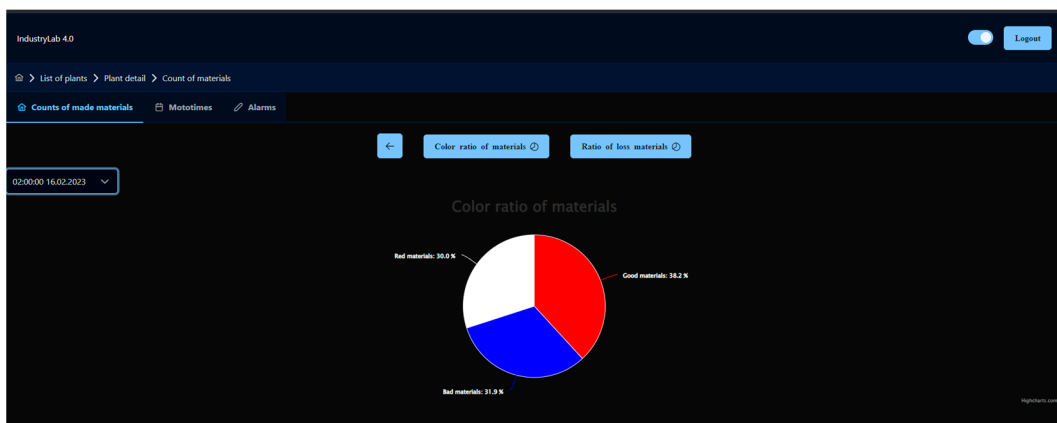


Figure 13. Pie chart of frontend application.

In practice, it is common for web applications to download data and generate content from them, such as data in a table. But in the HTML code the table expects the exact type of data, so separate components must be made for each type of data. In our case, there is one component for the list of all production lines, one component for the data table, one component for the line chart, and one for the pie chart. How individual components are generated depends on the data we enter into the configuration file. The configuration data for the production line model are compiled in the form of an object. In the entire configuration file, there is an array of multiple data object models from which the cards in the list are generated. The machine key contains the name of the card, the image key contains the path to the image, and the text key contains the accompanying text displayed below the name.

The detail key contains information about whether the given model contains data and what kind. In our case, the models can contain three types of data, alarms, engine hours, and the number of products. Therefore, the detail contains the keys' alarms, times, and counts. Each of the details contains the same data, which are also in the form of an object. In this object, the show key determines whether the model contains these data. If it does not contain them, the given button is not displayed in the model tab and it is not possible to click on the given data in the model detail submenu. The title key contains the name of the table, and the path key contains the path to the endpoint from which the data are requested. In the header key, all the columns that are displayed in the field, and each of the columns contains settings, such as its name, whether it can be sorted, and what type of data it contains. The data type determines what filtering would be applied to that column. Furthermore, the data key contains all the data that should be displayed in the table. The individual data contain their name in the data that are obtained from the

backend and in what way they should be transformed. In our case, we only transform dates to the dd.mm.yyyy format.

The last key is the graphs key, which determines how many and what kind of graphs for a given type of data are provided. The configuration data for the graphs are in the format of an array, according to which the graph buttons above the table are generated. Individual graphs represent objects in which the type key determines whether it is a line graph or a pie graph. The buttonText key contains the text displayed in the button above the table. The show key determines whether the graph will be displayed by default (in priority over the table) when displaying data. The unit key specifies the name of the x-axis and y-axis in the string format "axisX~axisY". The moreTimes key determines whether the select html element will be displayed with all of the time data. The data key determines which data will be displayed on the graph, and in the color key we can choose our own colors for individual data.

If we leave only an empty field in the color key, the colors will be determined by default. If we also leave the graphs key as an empty field, then the data will not contain any graphic representation of the data.

3.4. Development of a Mixed Reality Application

The important part of the smart platform development was the creation of an application for mixed reality, which we implemented in the environment of the Unity game engine. We worked specifically in version 2020.3.33f1, which is verified for compatibility with the MRTK software 2.8 development toolkit. This part is one of the key paper contributions, as it enables the detection of a real production model in mixed reality and contains an algorithm for a generic rendering of the visualization of individual production models. As part of the development, it was necessary to implement the following libraries and settings into the Unity application:

- Set up an application for MRTK development;
- Implement the MQTT library for communication with the local server;
- Implement the Vuforia library for object recognition;
- Create an algorithm for rendering the user interface.

Next, we describe the procedure for individual implementations of functionalities in the project.

3.4.1. MRTK

It was necessary to set the project up to be compatible for work with MRTK. Since the MRTK package is not in the package manager, the implementation had to be performed using the MS Mixed Reality Feature Tool, an external application. Using this application, we were able to import the MRTK package into the Unity project.

3.4.2. MQTT Communication

We used the M2MqttUnity library to implement MQTT communication in the Unity project. It provided us with basic functions, such as connecting and disconnecting to the MQTT broker, receiving data from the given session, and sending values to the MQTT sessions, which we specified. We laid the foundations of our communication algorithm on these functions.

The ConnectToMQTTServer function ensured the connection to the MQTT broker. The input data were a string in the format ipAdresa~port. Based on the received data, the function connected the application to the MQTT broker. The default value of the input string was 192.168.50.60~1883, which ensured that the function was called without entering data. Also, the algorithm contained a function for disconnecting communication, which was provided by the DisconnectFromMQTTServer function. This function also provided disconnection from all sessions that were active at the given time.

The function SendDataToMQTTServerByValue ensured the sending of data. In this case, the input variable was of the GameObject type. Each button that interacted with one

of the variables from the PLC device contained all the necessary data for communication. This also enabled the simple implementation of our algorithm, where the data it contains was not specified in advance. From the name of the object, the type of variable was found out, and it was taken from the global variable, which contained all the data from the PLC device. Subsequently, the data were sent to the MQTT broker.

The `SubscribeTopics` function provided us with a connection to the MQTT session. Functions like `StartSubscribePlant` and `GetJSONPlant` were built on top of this function. The `StartSubscribePlant` function allowed to connect to one of the models. This includes disconnecting from all sessions, clearing the current global variable, connecting to the `systemX_initializer`, the `systemX_reader` session, and requesting initialization data. The `GetJSONPlant` function, in turn, connects to the `systemX_json` session and requests user interface data for the given system.

Another important function was the `OnMessageArrivedHandler`. This decided the treatment of the data that the Unity application received from the MQTT broker. In this case, whether the data were user interface data or actual data from production models was recognized. We present both cases of data processing in detail below.

In the communication structure, the local server was the data intermediary between the PLC device and the Unity application. The local server therefore did not keep information about the state of all data, but these states had to be stored in the Unity application. However, since the Unity application is not specified for one type of production model, it was necessary to provide a dynamic variable with different types of data and different keys. Therefore, an important part of the proposed MQTT implementation is also the method of data storage in the Unity application.

The data storage resides in the `GlobalVariables` static class we created. This class contained the variable `MQTT_data` in the form of a dictionary, which was based on defining a key in a string format and also data in a string format. It was necessary to store the data in one form, as Unity has a very difficult time working with dynamic variables. The class also contained functions for handling global data. They were functions such as the getter, the setter, dictionary emptying, data format checking functions, or data format converting functions.

The getter and setter were functions based on returning or storing data based on a key. In the case of the setter, if the key was not found in the dictionary, a new key was created with the value we wanted to write. In the case of the getter, if the given key was not found in the dictionary, a null value was returned.

When initializing the new model of the production line, the data were filled with all the variables that the given PLC device contained, and when the data changed it simply updated the data to the current value. During MQTT communication, all data were received in a string format and therefore it was necessary to first convert them to the JSON format, from which the name of the variable and its value were then obtained. The global data could not be updated from the Unity application; every data change was sent to the PLC device, and, based on the change in the given variable in the PLC device, its current value was accepted. In this way, we ensured the reliable display of the current values of all variables from the PLC device, and it was impossible for the application to display the wrong value.

An example of communication can be seen in Figure 14. When the button is pressed, we send the value of the given variable using MQTT communication, which sets it in the PLC device. The local server records the change of the given variable and then sends it using MQTT communication to the Unity application, where the change in the value of the variable is stored in (from the point of view of the application) a global variable.

3.4.3. Vuforia

Vuforia [37] is an augmented reality (AR) software platform that enables developers to integrate digital elements into the physical world using mobile devices. The implementation of Vuforia in the Unity project allowed us to recognize 3D objects and track them in

real time. But this was preceded by the creation of 3D models of production lines, based on which Vuforia was able to recognize a real object.

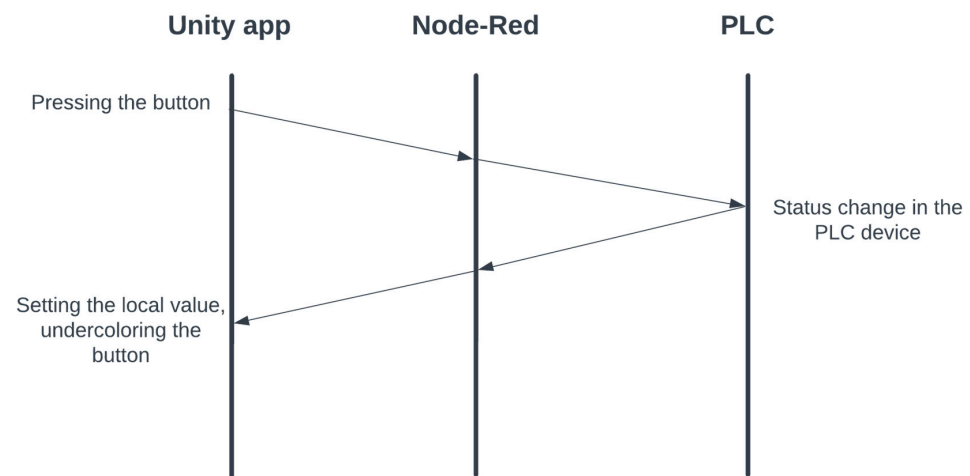


Figure 14. Communication within the entire project.

We tested several applications for creating 3D objects of production lines, such as Polycam, Scandy Pro, Metascan, Qlone, and MafiScan. Polycam was the most effective, providing us with the most authentic 3D models at no cost, with object size data that could be exported in common 3D model file types, e.g., in an .obj or .fbx format.

Using the Polycam application, we scanned 3D objects using a mobile device. In our case, it was an iOS mobile device, the iPhone 11 Pro Max. The device had a sufficiently high-quality camera, with which we captured more than 100 photos of the laboratory model of the production line. These photos were used by the application to create a 3D model of the corresponding production line and export it to an .fbx file. In this way, we created 3D models of the production lines. In Figures 15 and 16, we can see scanned 3D models created using the Polycam application.

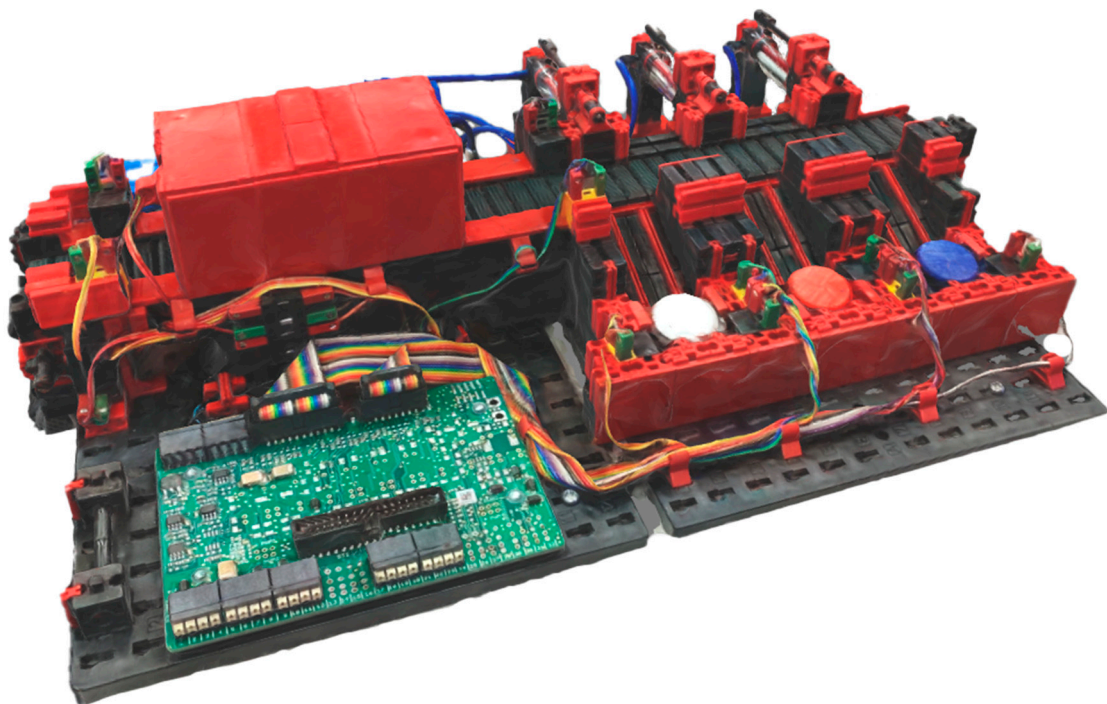


Figure 15. Scanned 3D model of the production line (model 2).

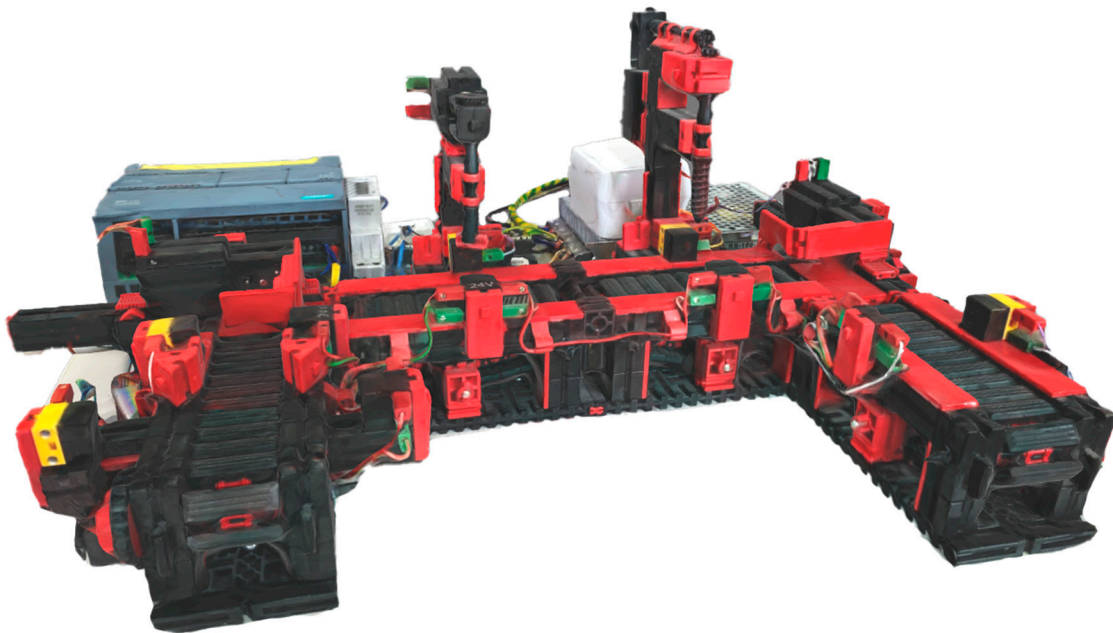


Figure 16. Scanned 3D model of the sorting line with color detection (model 1).

To detect real world objects, it was necessary to train the recognition of real models based on the obtained 3D models. This was enabled by the Vuforia Model Target Generator application. Based on the obtained 3D models, we created the SustavyPro database, which contained the data of the 3D models.

It was necessary to import the Vuforia Engine AR library into the Unity project. We implemented the trained database into the Unity project and created all the necessary objects for recognizing real objects. Here, all Vuforia objects had to be put into an empty object called VuforiaContent. If we skipped this step, the Vuforia library would not record objects at all. Objects System1 and System2 represent 3D objects, based on which real objects are recognized. Both objects required a connection to the database and the target model in the database. After connecting the target model, the dimensions of the object were automatically delivered.

Another necessary configuration was to set what should happen when objects are recognized and lost. We use the RecognitionObjects object for this, which provides functions that are performed in these events. When the object is recognized, the Unity project connects to all MQTT sessions, requests the local server for the JSON UI, and, after rendering the user environment, requests all the initialization data of the PLC device of the given system. When the object is lost, it is disconnected from all MQTT sessions and the currently generated user environment is deleted from the scene.

3.4.4. User Interface Rendering Algorithm

As already mentioned, the application renders the user interface of individual production lines based on the data defined by us—the JSON UI. The data reside in files on a local server where they can be changed at any time without the need to build a new version of the Unity application.

When receiving data from the local server, the MQTT session the data were received from is monitored. If the data were received from a session that carries a JSON string, the data will be treated according to the user interface rendering algorithm. This algorithm consists of the following steps:

- Reformatting data from a string format to a JSON format;
- Removing the previous user environment if it exists in the scene;
- The rendering of a new user environment;
- Connections to all MQTT sessions of the recognized production line;

- A request for initialization data of the current production line.

In the C# programming language, when formatting from a string we are expected to know in advance what data format the string will give us. In our case, we expect data in the form of an object that carries information about the name of the production model and its number. Based on the name of the model, the algorithm determines to which object in the scene it will render all the objects. Thus, if we accept the name PLC1, the data will be rendered into a 3D object named MenuPLC1. Each of the 3D models of production lines contains such a menu object, thanks to which we ensure that the user environment of the production line will always be located at the coordinates of the recognized production line. The production line number, in turn, determines which MQTT sessions the Unity application will connect to.

Furthermore, the object contains the field content, which represents the content of all the objects located in the user environment, for example, background objects that contain nested objects, such as texts, buttons, squares, or circles.

The background object contains information: the name of the 3D object, the background color, the visibility of the 3D object, the position and size of the 3D object, and the nested 3D object's content. We can set the background color to red, green, yellow, blue, gold, black, or white. The background color is represented as a renderer component, and the color spectrum is determined using the spectrum map. Every color that can be set on a 3D object was created in Photoshop. Since the user environment can contain several panels, the visibility of the 3D object determines which one will be active during rendering. When switching between panels, the currently displayed panel is deactivated and the panel to which we want to switch is activated. Nested 3D objects represent all objects that are displayed on the given panel. We list them below.

The key of the position object carries the data that each of the objects in the JSON UI representing the 3D object contains. It contains posY, posX, and posZ data, which are represented in a float format and determine the position of the 3D object depending on its parent 3D object. It also contains scaleX, scaleY, and scaleZ data, which determine the size in individual axes.

The key of the content object contains the array of nested objects that are located on the given panel. Depending on the type of 3D object, the data object can contain different data. Each data object representing a nested object contains data, such as the name, tag, gameObjectType, and position. Depending on the type specified by the gameObjectType key, the data object may additionally contain buttonProperties (in the case of a button), textProperties (in the case of text), or squadProperties (in the case of a circle and square).

Common data determine the basic properties of 3D objects. The name key represents the name of the 3D object, and the tag key specifies the tag to be assigned to the 3D object. According to the tag of the 3D object, which MQTT session the data will be sent to is determined. The gameObjectType key specifies what type of 3D object it is: in the case of a button, it contains the value button; in the case of text, it contains the value text; in the case of a square, it contains the value squad; and in the case of a circle, it contains the circle. The position key, as already mentioned, determines the position and dimension of the 3D object.

If it is a 3D text object data, it is supplemented with the text value (text), font size (fontSize), text color (textColor), text style (fontStyle), vertical alignment (verticalAlign), and horizontal alignment (horizontalAlign). In the case of the text type, the values can be normal, bold, or normal. The color of the text can be defined using the values black, white, red, blue, yellow, or green. The text can be aligned in the vertical plane using the values center, top, and bottom and in the horizontal plane using the values left, right, center, or flush. The flush value aligns the text from one edge to the other.

The 3D objects squad and circle are the simplest objects in the algorithm, and they only contain the color value, with which we can change the background color of the object. A circle, unlike a squad 3D object, is rendered with a 50% radius.

Button-type 3D objects are the most complex objects in our algorithm. In addition, the data contain the text that is displayed on the button and in the text label, to which the application reacts when controlled by the speech label. The setting of the icon of the button is enabled by the icon key, and the color of the button is set by the color key (with the same values as the background). The clickable key carries a Boolean value that specifies whether the button is clickable and a list of functions that will be performed when the button is pressed. If the button control is not enabled, the function list is ignored. Enabling button control determines whether the button will also include a visualization of the button outline in a 3D environment (Figure 17).

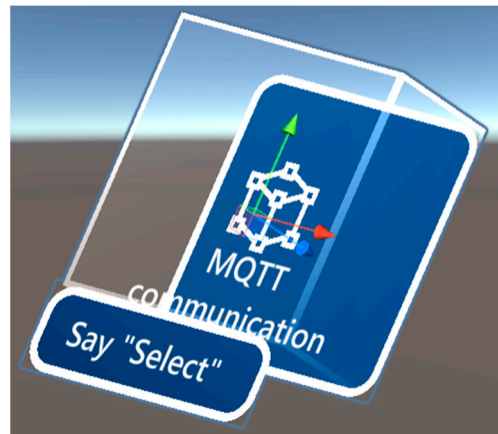


Figure 17. Three-dimensional visualization of the button.

Individual functions contain data, such as the function name (`functionName`), variable value, and variable type (`variableType`).

There are several functions to choose from:

- `SendDataToMQTTServerByValue`—sending data to the PLC device with value negation;
- `SendDataToMQTTServer`—fixed sending of data to the PLC device with a static value;
- `HandleVisibility`—switching the active panel;
- `HandleVisibilityByControl`—switching the active panel based on a variable;
- `HandleVisibilityByControlWithOtherValue`—the same function as the previous one, enriched by sending data to the PLC device.

Since some 3D objects, such as a button, contain multiple nested 3D objects, it would be too complicated to create new 3D objects in a script. Therefore, we created predefined 3D objects, called prefabs, in Unity. Based on the received data, we used specific prefab objects for buttons, texts, backgrounds, or squares/circles and adjusted their configuration according to the object type.

3.4.5. Appearance of the Application

Since the application was developed for the MS HoloLens 2 device, the appearance of the application was designed according to the MRTK design. The components from the MRTK templates library were used and modified according to our needs.

The application contains a main menu with the option of activating and deactivating communication with the MQTT broker, displaying the active production line, and debugging buttons for activating individual production lines. Since it is a mixed reality application, it was not possible to use common practices, like web or mobile applications. The menu is therefore realized by following the user with the possibility of opening and closing, as it occupies a large area when open. The menu is animated upon interaction.

Each of the lines provides multiple panels, and when a production line is recognized an initial panel is displayed with the option of redirecting to all panels that the production

lines contain. Redirection is performed by pressing the button. Each panel in this section is generated from the JSON UI.

For each of the control modes of the laboratory production models, a panel was designed according to Section 3.1. When switching control modes, the panels corresponding to the activated control mode are also switched automatically. In Figures 18 and 19, we can see panels of automatic modes at both production lines. In the manual mode, we can see the same layout of objects, with the difference that the buttons respond to pressing; in the automatic mode, the buttons are blocked.

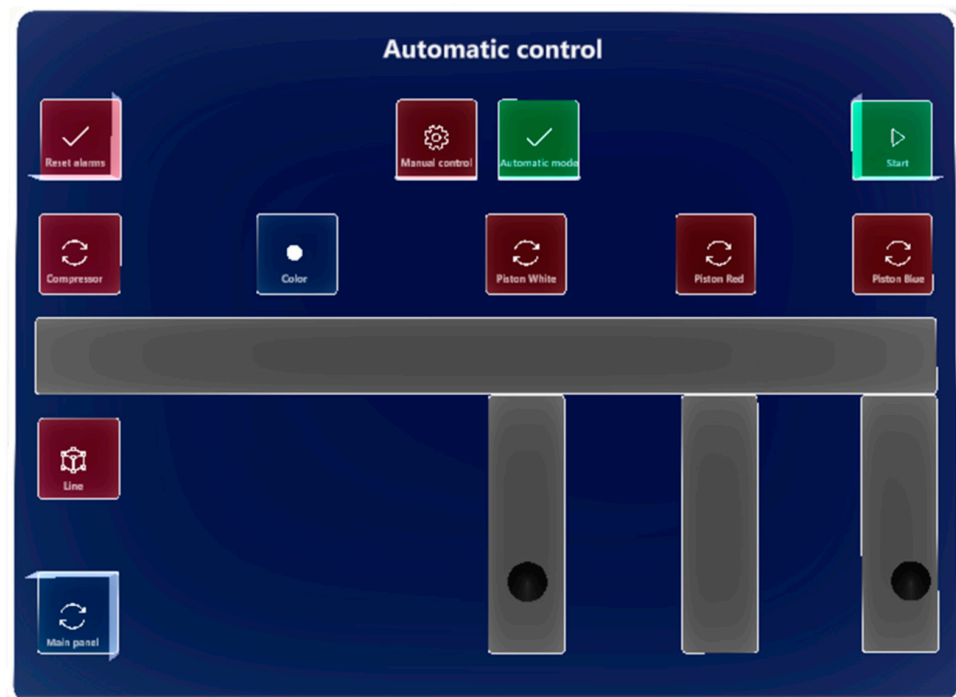


Figure 18. A panel with the possibility of monitoring and controlling the sorting line.

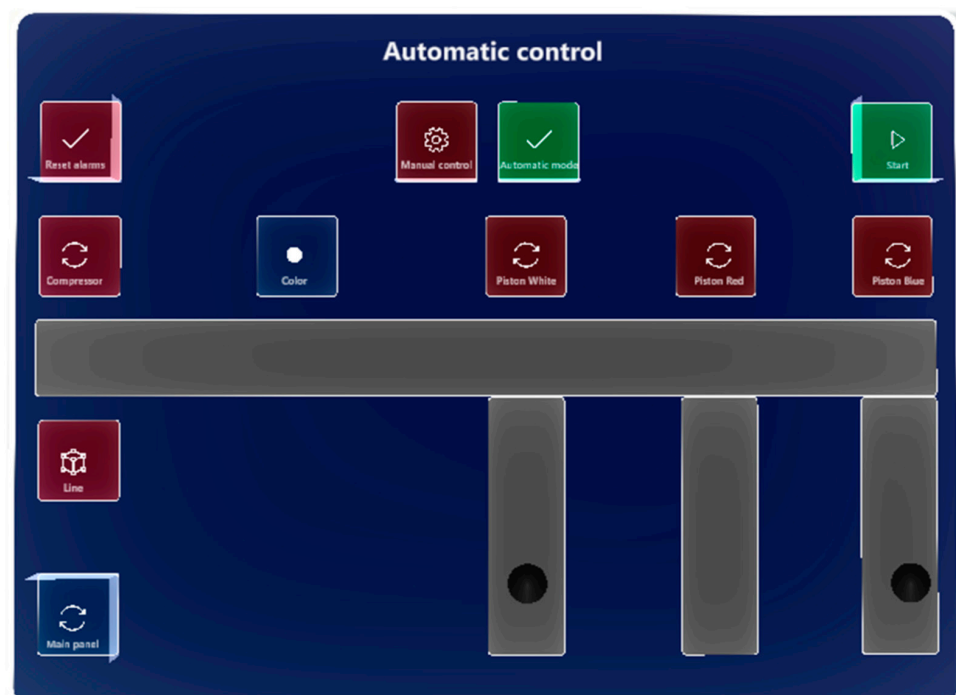


Figure 19. A panel with the possibility of monitoring and controlling the production line.

Once the line is recognized in the real world, the panels appear above the production line and hold that position throughout. With the loss of the detected production line, the panels are erased from the virtual reality scene. As the user moves, the panels tilt behind the user, thus ensuring a good viewing angle from any position of the user. Virtual panels placed above real production lines can be seen in Figures 20 and 21.



Figure 20. Production line detection in reality.

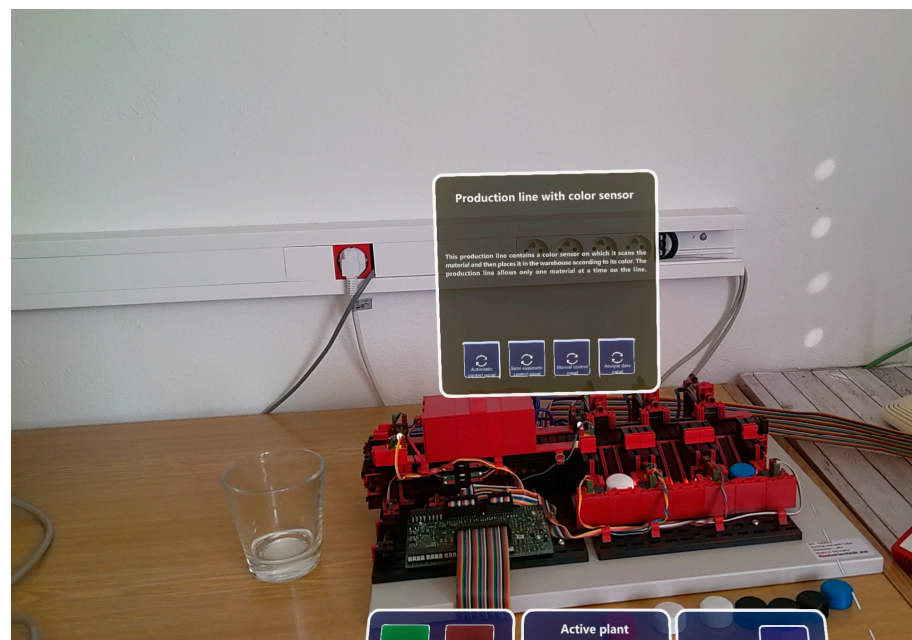


Figure 21. Sorting line detection in reality.

4. Conclusions

The use of augmented reality has significant potential to increase the operating and supervisory comfort in the production process. Monitoring and control data and their real time accessibility and display can improve production efficiency, predict machine maintenance, reduce machine downtime, and bring other benefits. The proposed platform

concept enables a new object to be added (device, machine, production line) with minimal effort, including its visualization. The proposed solution meets all requirements in terms of interoperability and scalability, due to the use of the OPC UA protocol and the use of configuration files for generating the content of individual devices and their functionalities in the application.

The contributions of the developed smart platform can be summarized as follows:

- 1 The design and implementation of an original smart platform for monitoring and controlling discrete event systems using intelligent technologies As part of the proposed solution, a software application system was designed and implemented to support the monitoring, diagnostics, and control of discrete event systems. The developed platform is based on modern smart technologies and concepts such as the Internet of Things, augmented reality, computer vision, and cloud computing. These are key technologies for the digitalization of production processes in practice and are included in the concept of Industry 4.0.
- 2 The implementation of a system for monitoring and controlling discrete event systems on a headset for augmented reality The application was implemented on the currently most advanced headset for augmented/mixed reality, Microsoft HoloLens 2, using the Mixed Reality Toolkit library. Conventional solutions use mobile devices, such as smartphones or tablets. The use of a headset thus brings many advantages, especially the fact that the operator's hands remain free when using the application system. We can consider the developed solution not only within the framework of Industry 4.0 but also as a human-centric solution, which is one of the pillars of the Industry 5.0 concept.
- 3 The use of industrial standards and industrial hardware components in the designed smart platform The application uses the industrial communication standard OPC UA and a PLC, which makes it different from other solutions that are often based on common communication protocols that can only be used in the consumer IoT and prototyping electronics, such as Arduino and the like.
- 4 The modification and expansion of the concept of definition schemes for the dynamic generation of a GUI in augmented reality, its implementation, and its verification significant result of the work is the modification and expansion of the concept of definition schemes for the dynamic generation of graphical user interfaces for augmented reality systems presented in the work. The modification and expansion consist of the use of the MS HoloLens 2 augmented reality visualization device and extended options for setting and visualizing objects.
- 5 The verification of the designed and implemented smart platform on two laboratory production lines The developed original smart platform has been tested and verified on two laboratory discrete event systems—production lines:
 - (a) A laboratory model of the production line with optical sensors, position sensors, conveyors, and two machining tools;
 - (b) A sorting line model for recognizing and sorting different workpieces based on color detection.

The proposed smart platform consists of a local server based on Node-RED, a cloud server with a database in the MS Azure environment, an application for displaying mixed reality in the Unity environment, and web applications in the Angular and Spring Boot 3 environments. The data can be displayed locally using mixed reality and can also be displayed remotely via a web browser.

The benefits of the work declared in the above points represent new trends in the development, research, and application of new procedures and solutions in the field of controlling and monitoring mechatronic systems. The results of the work can be used as a basis for further research.

Author Contributions: Conceptualization, F.Ž. and J.C.; methodology, J.C.; software, F.Ž.; validation, F.Ž., O.H., and E.K.; formal analysis, E.K. and D.R.; investigation, J.C.; resources, J.C., F.Ž. and D.R.; data curation, J.C. and E.K.; writing—original draft preparation, F.Ž. and J.C.; writing—review and

editing, J.C. and D.R.; visualization, F.Ž.; supervision, J.C. and D.R.; project administration, J.C.; funding acquisition, J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing is not applicable to this article.

Acknowledgments: The work has been supported by the Slovak Grant Agency VEGA 1/0107/22 and VEGA 1/0637/23, KEGA 039STU-4/2021, and Scientific Grant APVV-21-0125.

Conflicts of Interest: The authors declare no conflict of interest.

Glossary

IoT	Internet of Things
IEC	International Electrotechnical Commission
3D/4D	Three/Four Dimensional
SCADA	Supervisory Control and Data Acquisition
HMI	Human–Machine Interface
PLC	Programmable Logic Controller
HTML	HyperText Markup Language
MQTT	MQ Telemetry Transport
XR	Extended Reality
AR	Augmented Reality
VR	Virtual Reality
MR	Mixed Reality
SDK	Software Development Kit
REST	Representational State Transfer
API	Application Programming Interface
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator

References

1. Britannica, E. Industrial Revolution. Encyclopedia Britannica. 2021. Available online: <https://www.britannica.com/event/Industrial-Revolution> (accessed on 14 January 2022).
2. Zhou, K.; Liu, T.; Liang, L. From cyber-physical systems to Industry 4.0: Make future manufacturing become possible. *Int. J. Manuf. Res.* **2016**, *11*, 167. [\[CrossRef\]](#)
3. Huba, M.; Kozák, Š. From E-learning to Industry 4.0. In Proceedings of the 2016 International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 24–25 November 2016; pp. 103–108.
4. Ortiz, J.H.; Marroquin, W.G.; Cifuentes, L.Z. *Industry 4.0: Current Status and Future Trends*; InTech Open: Rijeka, Croatia, 2020. [\[CrossRef\]](#)
5. Rafique, W.; Qi, L.; Yaqoob, I.; Imran, M.; Rasool, R.U.; Dou, W. Complementing IoT services through software defined networking and edge computing: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1761–1804. [\[CrossRef\]](#)
6. Rosales, J.; Deshpande, S.; Anand, S. IIoT based Augmented Reality for Factory Data Collection and Visualization. *Procedia Manuf.* **2021**, *53*, 618–627. [\[CrossRef\]](#)
7. Baig, M.J.A.; Iqbal, M.T.; Jamil, M.; Khan, J. Design and implementation of an open-Source IoT and blockchain-based peer-to-peer energy trading platform using ESP32-S2, Node-Red and, MQTT protocol. *Energy Rep.* **2021**, *7*, 5733–5746. [\[CrossRef\]](#)
8. Mourtzis, D.; Angelopoulos, J.; Panopoulos, N. Collaborative manufacturing design: A mixed reality and cloud-based framework for part design. *Procedia CIRP* **2021**, *100*, 97–102. [\[CrossRef\]](#)
9. Frank, J.A.; Krishnamoorthy, S.P.; Kapila, V. Toward Mobile Mixed-Reality Interaction with Multi-Robot Systems. *IEEE Robot. Autom. Lett.* **2017**, *2*, 1901–1908. [\[CrossRef\]](#)
10. Kovacevic, N.; Meinzer, J.; Stark, R. Nutzerzentrierte Entwicklung einer ortsunabhängigen Maschinenabnahme mittels Augmented Reality. In *Entwerfen Entwickeln Erleben in Produktentwicklung und Design 2021, Proceedings of the Entwerfen Entwickeln Erleben—EEE2021, Dresden, Germany, 17–18 June 2021*; TUDpress: Dresden, Germany, 2021; pp. 417–429, ISBN 978-3-95908-450-5. [\[CrossRef\]](#)

11. Vysoký, L.J.S.A.H.; Dahl, R.; Priesum, J. Facebook, Device Interaction in Augmented Reality. USA Patent Klasifikovaný G06F3/04815—Interakcia s Trojrozmernými Prostrediami. U.S. Patent 20160274762, 16 March 2016.
12. Stark, E. Moderné Metódy Ovládania a Monitorovania Mechatronických Systémov s Využitím Počítačom Generovanej Reality. Diploma Thesis, FEI Ústav Automobilovej Mechatroniky, Bratislava, Slovakia, 2019.
13. Husinsky, M.; Schlager, A.; Jalaeefar, A.; Klimpfinger, S.; Schumach, M. Situated Visualization of IIoT Data on the Hololens 2. In Proceedings of the 2022 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), Christchurch, New Zealand, 12–16 March 2022; pp. 472–476. [\[CrossRef\]](#)
14. Wittenberg, C. Challenges for the human-machine interaction in times of digitization, CPS & IIoT, and artificial intelligence in production systems. *IFAC-PapersOnLine* **2022**, *55*, 114–119.
15. Žemla, F.; Cigánek, J. Digitization of event systems using intelligent technologies for Industry 4.0. In *ELITECH'22 [Elektronik Source]: 24th Conference of Doctoral Students, Bratislava, Slovakia, 1 June 2022*, 1st ed.; Spektrum STU: Bratislava, Slovakia, 2022; ISBN 978-80-227-5192-6.
16. Žemla, F.; Cigánek, J. Digitization of event systems in augmented reality using intelligent technologies. *Int. J. Inf. Technol. Appl.* **2021**, *10*, 25–36.
17. Khang, A.; Jadhav, B.; Birajdar, S. Industry Revolution 4.0: Workforce Competency Models and Designs. In *Designing Workforce Management Systems for Industry 4.0*; CRC Press: Boca Raton, FL, USA, 2023; pp. 11–34.
18. Henning, K. *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0*; Acatech: Munich, Germany, 2013.
19. Devezas, T.; Leitão, J.; Sarygulov, A. *Industry 4.0*; Springer: Basel, Switzerland, 2017.
20. Lemstra, M.A.M.S.; de Mesquita, M.A. Industry 4.0: A tertiary literature review. *Technol. Forecast. Soc. Chang.* **2023**, *186*, 122204. [\[CrossRef\]](#)
21. What Really Is the Difference between AR/MR/VR/XR? Available online: <https://medium.com/@northof41/what-really-is-the-difference-between-ar-mr-vr-xr-35bed1da1a4e> (accessed on 3 April 2023).
22. Interaction-Design. Augment Reality. Available online: <https://www.interaction-design.org/literature/topics/augmented-reality> (accessed on 3 April 2023).
23. Zagury-Orly, I.; Solinski, M.A.; Nguyen, L.H.; Young, M.; Drozdowski, V.; Bain, P.A.; Gantwerker, E.A. What is the current state of extended reality use in otolaryngology training? A scoping review. *Laryngoscope* **2023**, *133*, 227–234. [\[CrossRef\]](#) [\[PubMed\]](#)
24. Coelho, P.; Bessa, C.; Landeck, J.; Silva, C. Industry 5.0: The arising of a concept. *Procedia Comput. Sci.* **2023**, *217*, 1137–1144. [\[CrossRef\]](#)
25. Publication Office of the European Union. Enabling Technologies for Industry 5.0. Available online: <https://op.europa.eu/en/publication-detail/-/publication/8e5de100-2a1c-11eb-9d7e-01aa75ed71a1/language-en> (accessed on 3 April 2023).
26. Publication Office of the European Union. Industry 5.0. Available online: <https://op.europa.eu/en/publication-detail/-/publication/468a892a-5097-11eb-b59f-01aa75ed71a1/language-en> (accessed on 3 April 2023).
27. Siemens. TIA Portal. Available online: <https://www.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal.html> (accessed on 3 April 2023).
28. Technologies, U. Unity Real-Time Development Platform | 3D, 2D VR & AR Visualizations. Available online: <https://unity.com/> (accessed on 3 April 2023).
29. Frank. Siemens PLCs. Available online: <https://automationprimer.com/2014/03/16/siemens-plcs/> (accessed on 3 April 2023).
30. Siemens. HMI KTP 700 Basic. Available online: <https://support.industry.siemens.com/cs/pd/302298?pdtd=td&dl=en&lc=en-GB> (accessed on 3 April 2023).
31. Fischertechnik. Fischertechnik. Available online: <https://www.fischertechnik.de/de-de> (accessed on 3 April 2023).
32. Node-RED. Node-RED. Available online: <https://nodered.org/> (accessed on 3 April 2023).
33. Bigelow, S.J. Microsoft Azure. Available online: <https://searchcloudcomputing.techtarget.com/definition/Windows-Azure> (accessed on 3 April 2023).
34. Github. Github. Available online: <https://github.com/> (accessed on 3 April 2023).
35. Spring. Spring Boot. Available online: <https://spring.io/projects/spring-boot> (accessed on 3 April 2023).
36. Angular. Angular. Available online: <https://angular.io/> (accessed on 3 April 2023).
37. Vuforia. Vuforia. Available online: <https://developer.vuforia.com/> (accessed on 3 April 2023).

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.