computers

*Article*

# Implementation of Integrated Development Environment for Machine Vision-Based IEC 61131-3

**Sun Lim *, Un-Hyeong Ham and Seong-Min Han**

Intelligent Robotics Center, Korea Electronics Technology Institute, 655 Pyeongcheon-ro, Wonmi-gu, Bucheon-si 14502, Gyeonggi-do, Republic of Korea; uhham90@inu.ac.kr (U.-H.H.); tjdals2038@korea.ac.kr (S.-M.H.)
* Correspondence: sunishot@keti.re.kr; Tel.: +82-10-3540-9865

**Abstract:** IEC 61131-3 is an international standard for developing standardized software for automation and control systems. Machine vision systems are a prominent technology in the field of computer vision and are widely used in various industries, such as manufacturing, robotics, healthcare, and automotive, and are often combined with AI technologies. In industrial automation systems, software developed for defect detection or product classification typically involves separate systems for automation and machine vision programs, leading to increased system complexity and unnecessary resource wastage. To address these limitations, this study proposes an IEC 61131-3-based integrated development environment for programmable machine vision. We selected 11 APIs commonly used in machine vision systems, evaluated their functions in an IEC 61131-3 compliant development environment, and measured the performance of representative machine vision applications. This approach demonstrates the feasibility of developing PLC and machine vision programs within a single-controller system. We investigated the impact of controller performance on function execution.

**Keywords:** IEC 61131-3; machine vision; integrated development environment; industrial automation

**Citation:** Lim, S.; Ham, U.-H.; Han, S.-M. Implementation of Integrated Development Environment for Machine Vision-Based IEC 61131-3. *Computers* **2024**, *13*, 172. https://doi.org/10.3390/computers13070172

Academic Editor: Ananda Maiti

Received: 25 May 2024
Revised: 4 July 2024
Accepted: 8 July 2024
Published: 15 July 2024

## 1. Introduction

IEC 61131-3 provides programming standards for automation and control systems and plays a crucial role in smart factories. This international standard for standardized programming in automation and control systems defines standard programming languages, development environments, and program structures in the industrial automation field. In constructing smart factory systems, an IEC 61131-3-based integrated development environment is generally used as a key development tool for major devices ranging from level 0 to level 2 in the ISA 95 international standard hierarchy.

Machine vision systems are a leading technology in the field of computer vision and have been increasingly utilized in recent years in various domains, such as smart factories, robotics, and autonomous driving. Machine vision technology plays a critical role in enhancing productivity and efficiency in the smart factory sector. For instance, it can automatically detect the appearance, size, and shape of production items using cameras to identify defects or contaminants on product surfaces. Additionally, it can scan barcodes or QR codes to improve the tracking and management of products and raw materials, and automatically perform logistics and inventory management by communicating with systems such as MES. Thus, machine vision significantly contributes to the automation and optimization of production processes in smart factories, leading to improved productivity, quality control, and manufacturing capabilities.

However, machine vision systems and IEC 61131-3 integrated development environments are configured separately, leading to numerous problems. Each system operates on entirely different software and hardware platforms, making it difficult to exchange data and control signals between them. Additionally, a complex system configuration can increase setup and maintenance costs and result in low system efficiency owing to the allocation of different personnel and resources. Most importantly, in systems where real-time performance is crucial, the complexity arising from system decentralization can lead to critical errors or make it difficult to meet system requirements, thereby posing significant limitations.

To address these issues, this study proposes a design for programmable machine vision in an IEC 61131-3-based integrated development environment. The integrated development environment supports the development of key control equipment, such as cameras, I/O devices, and motor drives, using standardized programming languages.

The remainder of this paper is organized as follows. Section 2 provides background knowledge on machine vision technology and IEC 61131-3-supported integrated development environments and introduces related research. Section 3 presents the design of the machine vision integrated development environment. Section 4 discusses the experimental results and an evaluation of the proposed design. Finally, Section 5 concludes the study.

## 2. Background

### *2.1. IEC 61131-3 and Machine Vision System*

IEC 61131-3 is an international standard for industrial automation and control systems that defines the programming languages for these applications. This standard allows programmers and engineers to develop and maintain various types of automation equipment and systems, particularly programmable logic controller (PLC) systems. IEC 61131-3 defines the following five programming languages. (1) Ladder diagram (LD): a graphical language based on traditional relay logic that allows users to create logic programs similar to electrical circuit diagrams using functions, such as AND, OR, and NOR. (2) Sequential function chart (SFC): a graphical language for process control based on sequences, defining the order of operations through steps and transitions. (3) Function block diagram (FBD): a visual language that uses graphic blocks to represent functions or control systems. Each block performs a specific function, and the connections between blocks allow for the creation of complex logic. (4) Instruction list (IL): a low-level text-based language similar to an assembly language, although less commonly used in modern systems because of its increasing complexity. (5) Structured text (ST): a high-level text-based language similar to C that is suitable for implementing complex logic and algorithms. It supports conditional statements (IF, Switch) and loops (FOR, While), as well as variable types, such as arrays. IEC 61131-3 standardizes the development programs for automation systems using these five programming languages, enhances code reusability, and simplifies system maintenance.

A machine vision system is a technology that enables computers to process and interpret images by performing visual tasks traditionally performed by humans. Recently, they have been used in various industrial automation systems, including quality inspection, robotic task assistance, and medical image analysis. Typically, machine vision systems employ sensors, such as cameras, to capture images, which are then processed using software that performs filtering, post-processing for recognition enhancement, and other tasks to extract meaningful information desired by the user. Finally, based on this information, the system performs specific tasks or makes decisions based on predefined criteria. Machine vision technology has seen significant advancements in recent years, owing to the integration of AI and deep learning techniques. Although traditional image processing algorithms rely on methods such as edge detection, filtering, and histogram equalization, modern approaches leverage deep-learning technologies such as convolutional neural networks (CNNs) for image classification, object detection, and image segmentation. These advancements have broadened the application of machine vision technology in manufacturing and robotics to tasks such as quality inspection, product classification, and automated assembly. Additionally, they are widely used in autonomous driving for vehicle recognition, traffic flow analysis, and traffic signal recognition, as well as in the medical field for various imaging applications.

### *2.2. Related Work*

Various studies have been conducted to develop integrated development environments (IDEs) for automated programming that comply with the IEC 61131-3 standard and connect to different standards and software. Research has focused on integrated development approaches using the latest IEC 61131-3 standard for distributed control systems based on IEC 61499 [1]. Studies have explored IEC 61131-3-based programs that connect Matlab/Simulink simulation software, which is widely used in automotive and robotics, with PLC control programs [2], and integrated development environments that allow PLC programs to run on ROS/Gazebo [3,4].

There has also been significant research on providing vendor-independent IEC 61131-3-compliant open-source integrated development environments [5,6]. Research aimed at defining purpose-specific software standards and development environments based on IEC 61131-3 has also been conducted [7–9].

Various studies have examined the overhead of the IEC 61131-3-based software in terms of controller performance and improvements in multicore environments. Research has been conducted on methodologies for efficiently writing and executing IEC 61131-3 software in low-spec embedded hardware environments [10–12]. Additionally, to overcome the limited performance of single-core embedded controllers, studies have explored the use of multicore systems [13–15] and parallelism techniques in IEC 61131-3 software that considers multicore architectures [16].

Research has been conducted on standard programming environments and application software to develop software related to machine vision applicable to various automation processes. Studies have focused on embedded controller-based machine vision systems for defect detection in the automated assembly of PCB boards [17,18]. Additionally, research has been conducted on vision systems based on embedded controllers in autonomous driving environments such as mobile robots [19]. Further studies have explored methodologies for developing and implementing machine vision software in environments using simulation or standardized middleware software [20,21]. Research has also been conducted on integrated system environments that interact with various pieces of equipment used in automation processes. Examples include the development of precision machining technology through the integration of CNC machines and vision systems for 3D machining [22] and the development of precise path planning algorithms through cooperative systems of measuring equipment and machine vision systems [23].

## 3. Machine Vision System

### *3.1. Integrated Machine Vision System Based on IEC 61131-3*

In conventional automated process systems, vision systems, and industrial controllers are configured separately. Industrial controllers handle real-time tasks such as motion control, sensor management, and robot control, whereas vision systems consist of computers equipped with cameras and image processing capabilities to execute image processing applications. The separation of control systems within automated processes poses challenges in terms of integrated control and management. To address these issues, this study proposes an integrated development environment in which real-time control and machine vision applications can be executed on a single industrial controller, as shown in Figure 1. The industrial controller manages all the subordinate elements, including the equipment within the automated process and cameras for the machine vision system. The cameras can be connected to an industrial controller through non-real-time interfaces, such as Ethernet switches, USB, and serial communication. In addition, real-time industrial communication via a fieldbus can connect the controller to robots, sensors, and motor drives. Using a unified development environment, users or process developers can develop automated process programs and download them to an industrial controller, thereby enabling the development of an integrated automated process system.

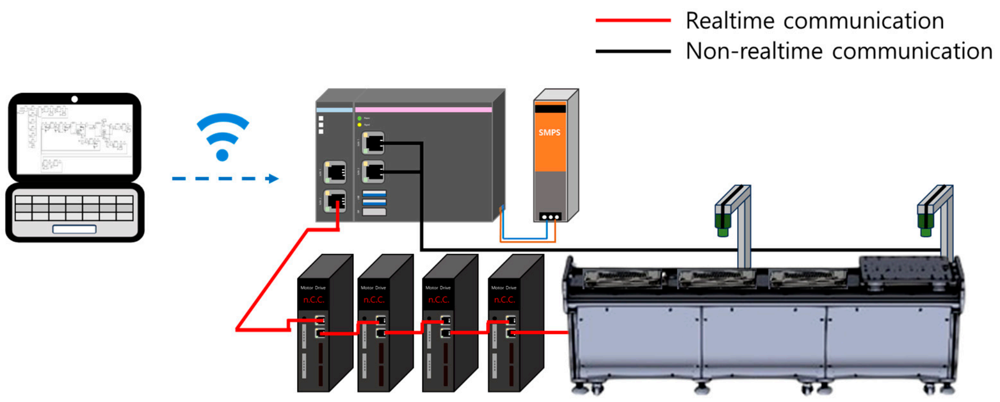


**Figure 1.** Integrated machine vision system.

To develop an integrated machine vision system, the industrial controller must have a software architecture capable of configuring real-time and non-real-time domains, as illustrated in Figure 2. Traditionally, applications that operate in real time, such as motor drives, sensors, and robots, must be based on a real-time operating system (RTOS) or controller. However, machine vision applications are typically developed to operate within a non-real-time domain because predicting execution time and prioritization are not critical factors. Considering these characteristics, the industrial controller should support both real-time and non-real-time domains and provide a programming environment that satisfies the real-time requirements. To address this issue, as shown in Figure 2, an industrial controller was configured with an OS based on an RT-patched Linux kernel. The industrial fieldbus can operate within RT-process applications, whereas all libraries and applications related to machine vision are designed to run in the form of non-real-time Linux processes.

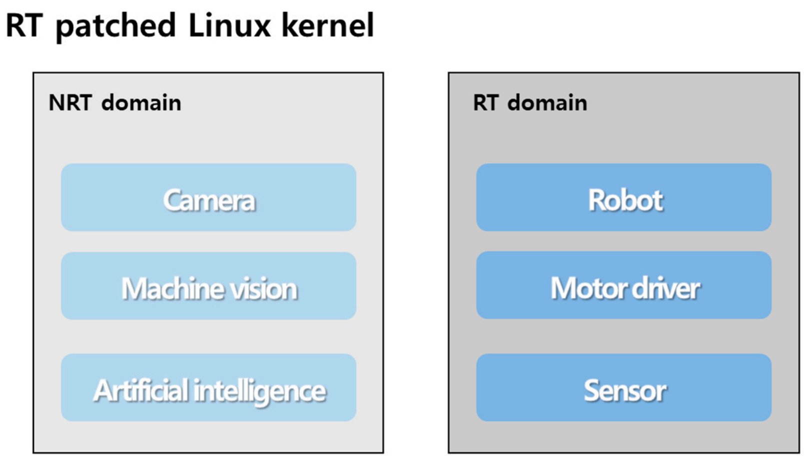


**Figure 2.** Real-time and non-real-time domain configuration.

### *3.2. Function Procedure Call for Machine Vision*

One of the key contributions of the proposed integrated machine vision system is its support for a programming environment based on the IEC 61131-3 international standard. Traditional machine vision systems have been developed separately according to user specifications, such as camera manufacturers and software development languages, leading to low software reusability. To address this issue, we developed a hardware-independent integrated development environment for machine vision systems based on IEC 61131-3 international standard language.

The industrial controller of an integrated machine vision system enables the development of various automated process applications. For example, programs for controlling a 6-axis articulated robot or PLC can be developed and structured based on the IEC 61131-3 international standard language. Applications developed using the IEC 61131-3 standard language are assumed to operate in the real-time domain. This is because the IEC 61131-3 international standard is defined as a programming language for PLC systems that are typically developed as real-time systems. However, software related to machine vision, such as camera control and image processing libraries, are developed for non-real-time applications. This is because predicting the execution time and determining the priorities between processes or tasks in machine vision applications is difficult.

Considering the characteristics of existing systems, this study proposes a function procedure call (FPC) to support IEC 61131-3-based machine vision programming. Libraries were structured to enable real-time domain applications to be developed using the IEC 61131-3 programming language without any modifications. However, functions related to camera control, image processing, and AI machine vision are structured as libraries in both the real-time and non-real-time domains. In the real-time domain, the machine vision library consists of only input and output parameters, without a separate execution code. In the non-real-time domain, the machine vision library is composed of actual machine vision execution codes.

The workflow of the FPC for machine vision programs is shown in Figure 3. At any arbitrary point, the machine vision function first captures the input parameters. It then transmits the values to a buffer to monitor periodic function calls within the non-real-time domain through the shared memory. When a function execution request is confirmed at a specific point, the corresponding function is immediately called and the input parameter values are passed. After the called function is executed, its result is periodically written to the monitoring buffer of the execution result of the called function within the real-time domain of shared memory. Finally, the called machine vision function writes the output value to the final variable and ends its execution.

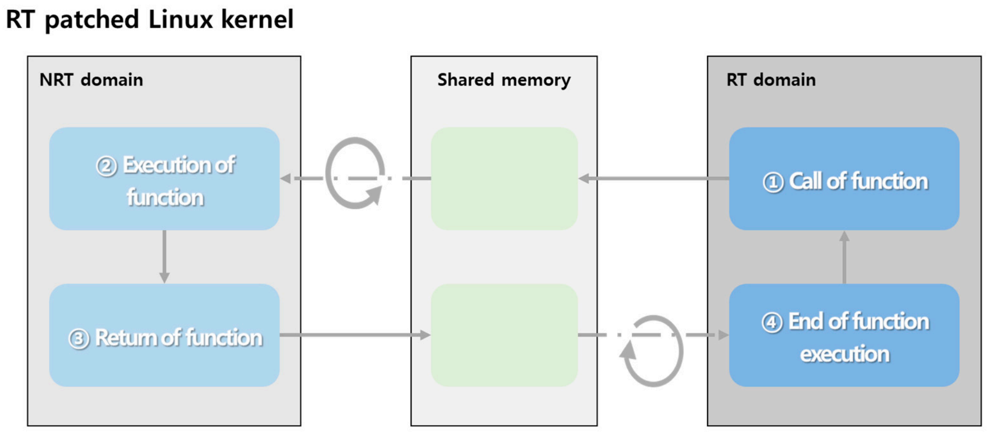


**Figure 3.** Workflow of a function procedure call.

### 3.3. IEC 61131-3 FBD for Machine Vison

Eleven types of functional blocks were designed to validate the IEC 61131-3-based integrated machine vision system. Machine vision software primarily comprises camera-related function blocks, image-processing function blocks, and AI-based vision software, as listed in Table 1. Function blocks starting with CAM_ represent camera control function blocks, defining functions such as establishing and terminating connections between the integrated controller and camera, setting and reading camera parameters, and acquiring images. The image processing library and AI-based vision function blocks were designed considering the essential functions commonly used in industrial machine vision systems. In addition, five types of image-processing function blocks and three types of AI-based vision function blocks were proposed.

**Table 1.** Function block for machine vision.

| Function Block Name | Description |
|---|---|
| CAM_OPEN | Enable the camera connection |
| CAM_CLOSE | Disenable the camera connection |
| CAM_GRAB | Take a camera video to get an image |
| CAM_SETPARAMETER | Set the parameters of the camera for taking images |
| CAM_GETPARAMETER | Get the parameters of the camera for taking images |
| Image_ScanBarcode | Read the barcode in the image |
| Image_ScanQrcode | Read the qrcode in the image |
| Image_TemplateMatching | Similarity test with genuine product |
| Image_Blob | Spot area detection using brightness and color |
| Image_Measurement | Measure the distance between objects |
| AI_Classification | Perform classification on learned objects |

Figure 4 illustrates an example of a classification function block designed in compliance with the IEC 61131-3 FBD standard language among the proposed 11 types of function blocks. The FBD language is primarily composed of input and output parameters, with each parameter having attributes defined by IEC 61131-3 standard variable types (BOOL, INT, REAL, etc.). The left and right sides of the function block represent input and output parameters, respectively. Figure 5 shows the FBD design for the 11 types of function blocks listed in Table 1. Each function block typically includes an executable input parameter for its start signal. It also includes output parameters, such as execution completion status (Done), execution status (Busy), and error status (Error, ErrorID). Additionally, separate input and output parameters are defined for each function block based on its specific functionality, which is designed to maximize the versatility of the function block.

The Beremiz open-source software was used to implement the proposed IEC 61131-3-based machine vision function blocks. Beremiz is an integrated development environment programming software based on the IEC 61131-3 standard language. Internally, it employs the MatIEC and GCC compilers to generate C code from the PLC code written by the user, enabling execution, building, and debugging. Beremiz allows the design of function block layouts in XML and supports users in programming in the editor through an XML parser. Figure 6 on the left side presents an example code of the FBD classification written in this XML language among the 11 types of function blocks.

On the right side of Figure 6, an example of code from the FPC in Section 3.2 executed in the real-time domain is shown. When the execution is signaled as TRUE, the function block captures the input parameter values and transmits them to the shared memory buffer of the non-real-time domain. It then monitors the shared memory buffer of the output parameters per cycle to verify the completion status of the invoked function, after which it reads the value and writes it to the output parameters.

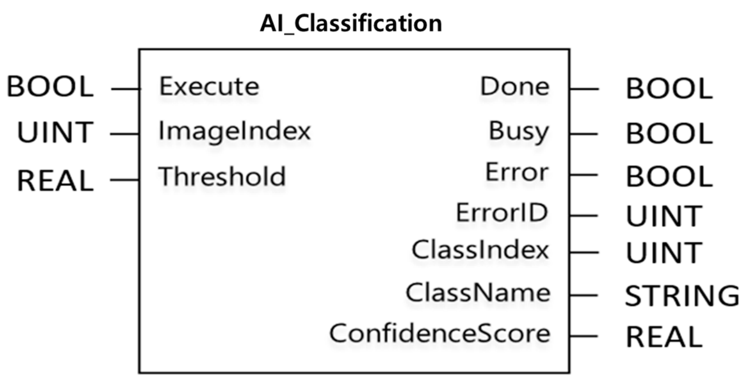


**Figure 4.** Example of AI_Classification function block.

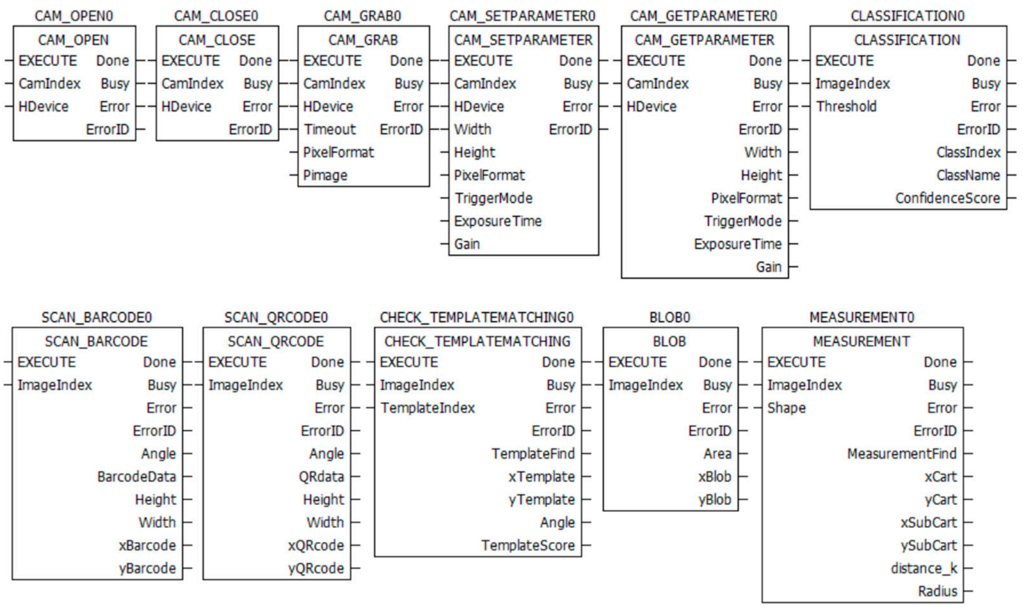
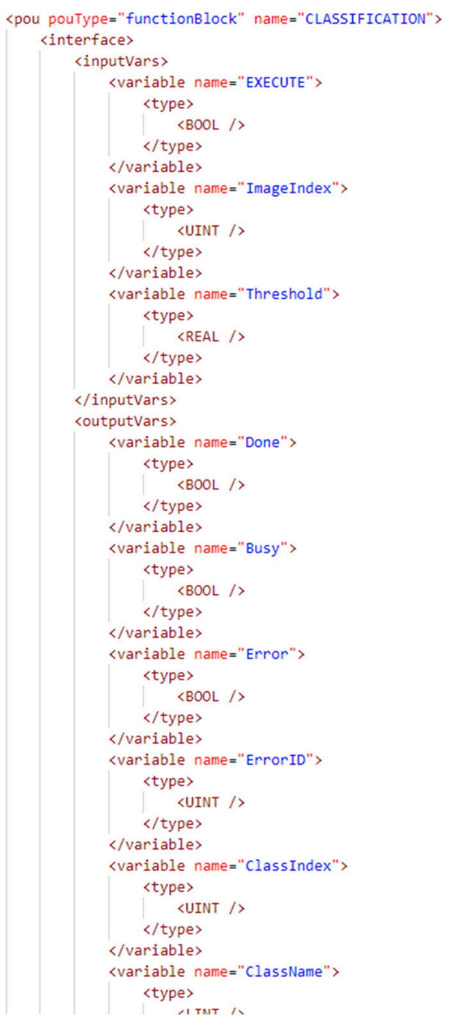
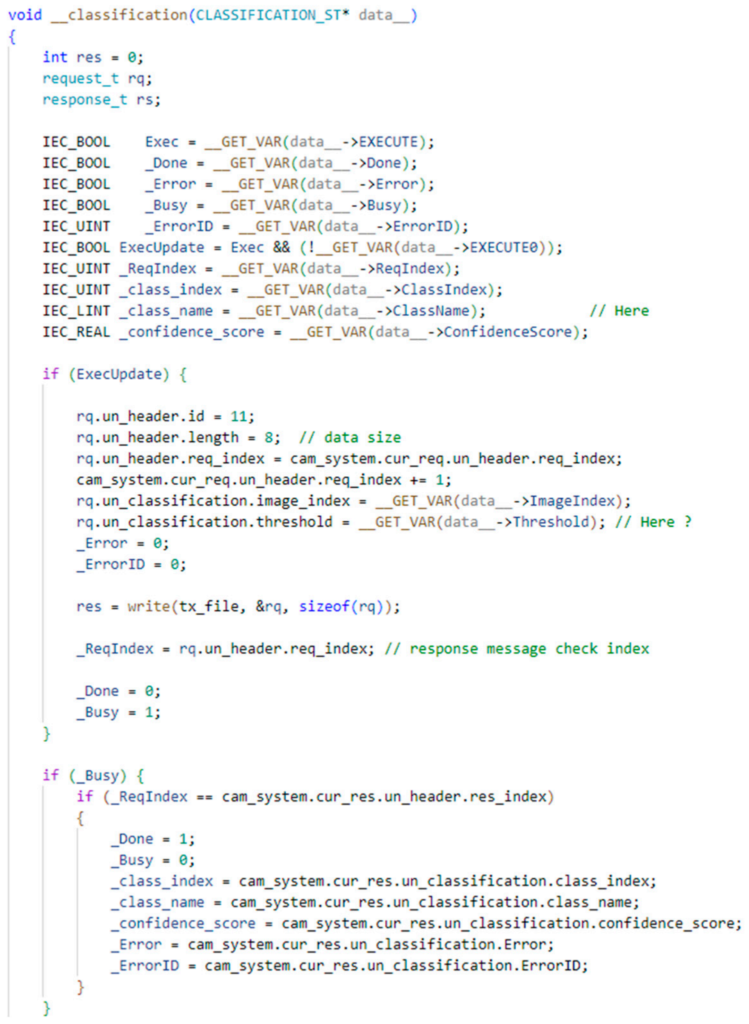


**Figure 5.** Function block list.



```xml
<pou pouType="functionBlock" name="CLASSIFICATION">
    <interface>
        <inputVars>
            <variable name="EXECUTE">
                <type>
                    <BOOL />
                </type>
            </variable>
            <variable name="ImageIndex">
                <type>
                    <UINT />
                </type>
            </variable>
            <variable name="Threshold">
                <type>
                    <REAL />
                </type>
            </variable>
        </inputVars>
        <outputVars>
            <variable name="Done">
                <type>
                    <BOOL />
                </type>
            </variable>
            <variable name="Busy">
                <type>
                    <BOOL />
                </type>
            </variable>
            <variable name="Error">
                <type>
                    <BOOL />
                </type>
            </variable>
            <variable name="ErrorID">
                <type>
                    <UINT />
                </type>
            </variable>
            <variable name="ClassIndex">
                <type>
                    <UINT />
                </type>
            </variable>
            <variable name="ClassName">
                <type>
                    <LINT />
```

Example of XML representation



```c
void __classification(CLASSIFICATION_ST* data__)
{
    int res = 0;
    request_t rq;
    response_t rs;

    IEC_BOOL    Exec = __GET_VAR(data__->EXECUTE);
    IEC_BOOL    _Done = __GET_VAR(data__->Done);
    IEC_BOOL    _Error = __GET_VAR(data__->Error);
    IEC_BOOL    _Busy = __GET_VAR(data__->Busy);
    IEC_UINT    _ErrorID = __GET_VAR(data__->ErrorID);
    IEC_BOOL ExecUpdate = Exec && (!__GET_VAR(data__->EXECUTE0));
    IEC_UINT _ReqIndex = __GET_VAR(data__->ReqIndex);
    IEC_UINT _class_index = __GET_VAR(data__->ClassIndex);
    IEC_LINT _class_name = __GET_VAR(data__->ClassName);            // Here
    IEC_REAL _confidence_score = __GET_VAR(data__->ConfidenceScore);

    if (ExecUpdate) {

        rq.un_header.id = 11;
        rq.un_header.length = 8;  // data size
        rq.un_header.req_index = cam_system.cur_req.un_header.req_index;
        cam_system.cur_req.un_header.req_index += 1;
        rq.un_classification.image_index = __GET_VAR(data__->ImageIndex);
        rq.un_classification.threshold = __GET_VAR(data__->Threshold); // Here ?
        _Error = 0;
        _ErrorID = 0;

        res = write(tx_file, &rq, sizeof(rq));

        _ReqIndex = rq.un_header.req_index; // response message check index

        _Done = 0;
        _Busy = 1;
    }

    if (_Busy) {
        if (_ReqIndex == cam_system.cur_res.un_header.res_index)
        {
            _Done = 1;
            _Busy = 0;
            _class_index = cam_system.cur_res.un_classification.class_index;
            _class_name = cam_system.cur_res.un_classification.class_name;
            _confidence_score = cam_system.cur_res.un_classification.confidence_score;
            _Error = cam_system.cur_res.un_classification.Error;
            _ErrorID = cam_system.cur_res.un_classification.ErrorID;
        }
    }
```

Example of FPC real-time domain operation

**Figure 6.** Example of XML representation and FPC real-time domain operation for classification.

## 4. Performance Evaluation

In this section, we conduct a performance evaluation of the 11 function blocks based on the IEC 61131-3 international standard proposed in Section 3. This study aims to compare the overhead of 11 function blocks according to controller specifications and evaluate the performance of user programs written in the actual IEC 61131-3 standard.

To conduct the performance evaluation, an experimental environment was established, as depicted in Figure 7, and the measurements were performed accordingly.The user IDE is a development environment capable of programming artificial intelligence machine vision based on the IEC 61131-3 standard, which is executable on the integrated controller. The detailed specifications are listed in Table 2. In addition, Beckhoff EtherCAT equipment and an oscilloscope were set up to measure the execution time of the function blocks. The specifications of the integrated controller are listed in Table 3.

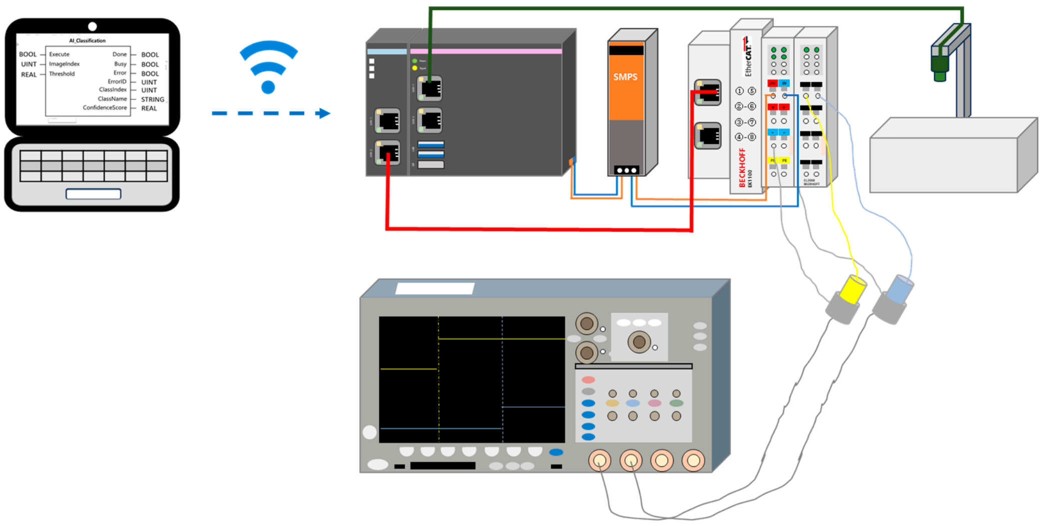


**Figure 7.** Experimental setup.

**Table 2.** Specifications of the integrated development environment.

| Function Block Name | Description |
|---|---|
| OS | Window 11 Pro |
| CPU | i9-13900H(P-Core upto 5.4 GHz/E-core upto 4.1 GHz/ 14Core/6 P + 8 E) |
| GPU | NVIDIA GeForce RTX 4070 8 GB GDDR6 |
| RAM | 32 GB (2 × 16 GB) DDR5 MEMORY |

**Table 3.** Specifications of integrated controller.

| IPC | Description |
|---|---|
| Processor | Intel Atom® x7-E3950 Processor |
| Memory | 8 GB LPDDR4-2400 Mhz |
| SATA | mSATA 1 slot (empty) |
| EtherNet | 1Gbe x2 (i210 x 2) |
| Power Input | 24 V |
| OS | Ubuntu 18.04.5 |
| Linux Kernel | 4.19.295 rt-129 |
| Software | openCV 4.5.3 Tensorflow 2.4.1 Pyro 3.9.1 |

This study establishes an environment utilizing the open-source Beremiz software version 1.2 based on open-source principles for industrial automation and control system development, along with an EtherCAT Master-based integrated controller. Beremiz serves as an open-source integrated development environment capable of programming based on the IEC 61131-3 standard, with the advantage of easily allowing the addition of user function blocks. EtherCAT is an essential Ethernet-based fieldbus widely used in PLC systems and is a core element in process automation technology, along with machine vision systems.

In this section, we measure the execution times of the 11 function blocks developed based on the IEC 61131-3 international standard PLC language proposed in Section 3. The measurement method involves using an oscilloscope to measure the changes in electrical signals for each function block and calculating and analyzing the average, variance, standard deviation, and maximum and minimum values. The experimental setup for the measurement included an industrial controller where the function blocks were operated, a laptop for the integrated development environment capable of remote operation, a Beckhoff module to generate electrical signal changes at the start and end of each function block, and an oscilloscope for measurement, each configured with one unit. The specifications of the laptop with the integrated development environment are listed in Table 2. The Beckhoff module utilized the EK1100 model with an EL2008 I/O module attached, and the oscilloscope used was the Tektronix MSO 4054 model.

### *4.1. Machine Vision Function Blocks for IEC 61131-3*

This experiment involved operating 11 function blocks using the industrial controller, as depicted in Table 3, and measuring the execution time through communication between the operating system, the application program, and the function blocks. Each function block was executed 30 times to measure the execution time. The experimental results, including the mean and standard deviation, are presented in Table 4.

**Table 4.** Measurement of API for 11 types of low-spec industrial controllers.

| Function Block | Mean | Standard Deviation |
|---|---|---|
| CAM_OPEN | 2.640 s | 0.105451411 |
| CAM_CLOSE | 2.647 s | 0.048383882 |
| CAM_GRAB | 0.770 s | 4.722993401 |
| CAM_SETPARAMETER | 0.506 s | 5.299895177 |
| CAM_GETPARAMETER | 0.473 s | 51.72345266 |
| Image_ScanBarcode | 0.352 s | 150.4945957 |
| Image_ScanQrcode | 0.433 s | 154.2946100 |
| Image_TemplateMatching | 14.903 s | 0.181628436 |
| Image_Blob | 0.176 s | 47.65953513 |
| Image_Measurement | 0.210 s | 27.09112196 |
| AI_Classification | 11.336 s | 0.116856988 |

Among the 11 functional blocks, with the exception of Template Matching and Classification, relatively short execution times were observed. Except for CAM_OPEN and CAM_CLOSE, execution times of less than one second were recorded. CAM_OPEN and CAM_CLOSE exhibited higher execution times because of the socket buffer creation and removal for communication between the integrated controller and the camera. However, other camera-related function blocks utilized existing sockets, resulting in shorter execution times. After conducting experiments and analyses on the 11 function blocks, it was found that Template Matching had the longest execution time, approximately 14.9 s, ranking first, whereas Classification ranked second with an execution time of approximately 11.3 s. Their standard deviations, ranging from 0.10 to 0.20, were relatively small, indicating reliability. However, among the fastest-operating blocks, Blob had a relatively high standard deviation. This is attributed to instances where it operated faster than the average, such as 50.6, 79.6, and 90.4 ms, within the typical range of 170 to 250 ms.

### *4.2. Analysis of Effect According to Controller Performance*

Experiments were conducted on four function blocks in a new integrated controller environment to measure the overhead of the machine vision function blocks based on the controller performance. In the previous experiment, it was observed that the Template Matching and Classification function blocks required significantly more execution time than the other function blocks. However, these long execution times pose limitations that must be addressed in future industrial automated system designs and applications. Therefore, to address this issue, a relatively high-performance controller hardware was selected, and the same experimental environment was set up. The specifications of the controller are listed in Table 5.

**Table 5.** Specifications of the integrated high-performance controller.

| IPC | Description |
|---|---|
| Processor | Intel® Q370 Platform Controller Hub |
| Memory | 32 GB DDR4 (Normal Temp 0 °C~+70 °C) |
| SATA | internal SATA port for 2.5″ HDD/SSD |
| EtherNet | Gigabit Ethernet ports by I219 and 5x I210 |
| Power Input | 160 W AC/DC power adapter 20 V/8 A |
| OS | Ubuntu 18.04.5 |
| Linux Kernel | 4.19.295 rt-129 |
| Software | openCV 4.5.3 Tensorflow 2.4.1 Pyro 3.9.1 |

To observe how the execution time varies according to CPU performance, the performance of Template Matching and Classification, which took the longest time to execute, and Blob, which executed the fastest, were compared and analyzed. The results of measuring the execution times of the four function blocks on the high-performance industrial controllers are listed in Table 6. Additionally, to analyze the standard deviation values, the execution time for each run was graphically represented in Figures 8–11.

**Table 6.** Measurement of execution times for Template Matching, Blob, Measurement, and Classification.

| Function Block | Low IPC | High IPC |
|---|---|---|
| Image_TemplateMatching | 14.903 s | 3.341 s |
| Image_Blob | 0.176 s | 0.056 s |
| Image_Measurement | 0.210 s | 0.112 s |
| AI_Classification | 11.336 s | 10.560 s |

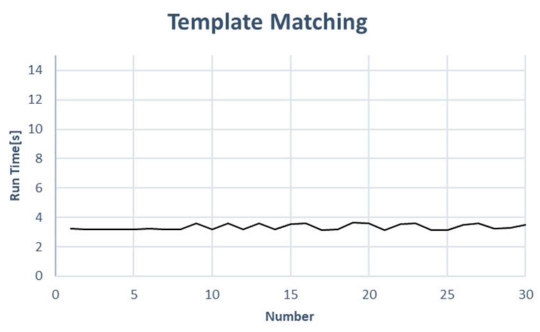


**Figure 8.** Execution time of Template Matching.

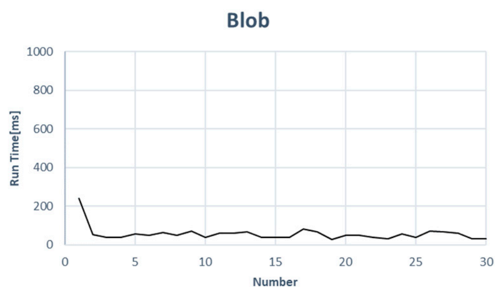


**Figure 9.** Execution time of Blob.

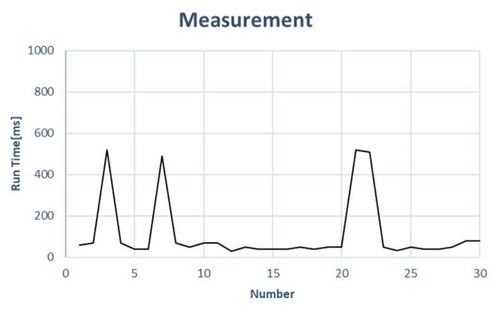


**Figure 10.** Execution time of Measurement.

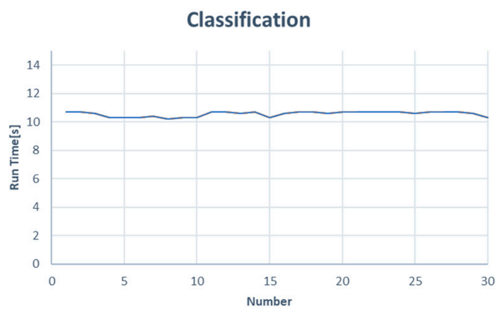


**Figure 11.** Execution time of Classification.

The comparative analysis revealed that for image processing function blocks requiring significant CPU operations, there was an approximately 2 to 4-fold performance difference between the high-spec and low-spec controllers. However, for AI-based machine vision requiring parallel processing, the performance difference between the low-spec and high-spec controllers improved by approximately 1.2 s in Table 7. This improvement, although notable, is not as significant as that observed in the simple image-processing function blocks. In the case of AI-based machine vision function blocks such as classification, the serial calculation process of the CPU limits efficiency gains, which necessitates processing numerous matrices in parallel. Therefore, it was concluded that when using AI FBDs in industrial controllers, attaching specialized parallel processing devices, such as GPUs or NPUs, and conducting optimizations, such as network lightweighting, are necessary to achieve significant performance improvements.

**Table 7.** Measurement of API for 11 high-end industrial controllers.

| Function Block | Mean | Standard Deviation |
|---|---|---|
| Image_TemplateMatching | 3.341 s | 0.197384284 |
| Image_Blob | 0.056 s | 36.8274656 |
| Image_Measurement | 0.112 s | 156.2497431 |
| AI_Classification | 10.560 s | 0.176257388 |

### 4.3. Use-Case of IEC 61131-3 Based Machine Vision Application

This study aims to operate automation and machine vision programs within a unified development environment to minimize unnecessary resource wastage and efficiently operate systems. Recent industrial automation systems have utilized AI-based classification algorithms for high-precision defect detection and product classification. Therefore, experiments were conducted by configuring scenarios in which the target objects were captured with cameras, detected, and discriminated using AI-based classification algorithms, as shown in Figure 12.

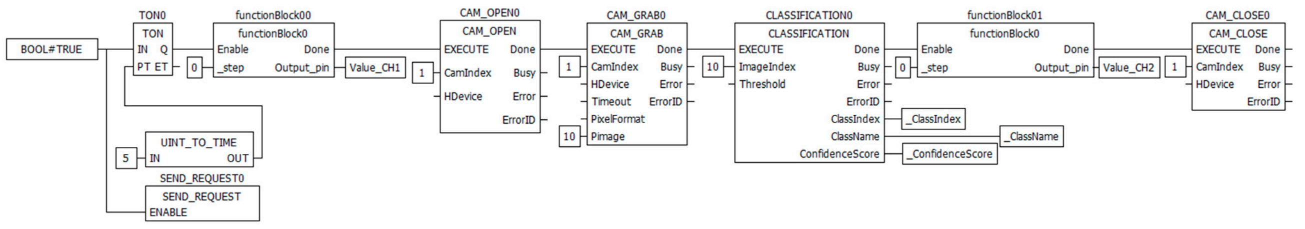


**Figure 12.** Example of IEC 61131-3 based classification application.

In this experiment, similar to the previous ones, the average and standard deviation of the execution time were calculated for 30 runs. in Figure 13. The shortest execution time was 12.5 s, whereas the longest was 17.1 s. The average time was 13.8 s with a standard deviation of approximately 0.8. Because the sum of the average execution times of the camera and each AI function block used in the experiment was less than or equal to the linearly added value, it was possible to anticipate the expected operation time during the program configuration.

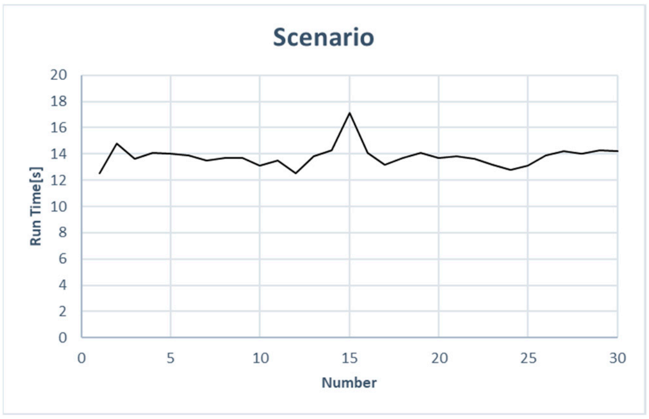


**Figure 13.** Execution time of scenario.

## 5. Conclusions

In this study, we propose the idea of developing vision programs commonly used in existing automation process systems in an IEC 61131-3 supported integrated development environment. To achieve this, an industrial controller was configured as a software architecture capable of real-time and non-real-time domain configurations, and a function procedure-call structure for machine vision function processing was developed. This was designed to minimize the impact on programs running in the existing real-time domain and maintain the characteristics of existing machine vision systems to develop a reliable integrated development environment. For the performance evaluation of the developed system, 11 commonly used APIs in machine vision systems were selected, and their designs were developed to be programmable in the IEC 61131-3 FBD programming language. These were programmed, and the performance was evaluated using Beremiz open source software to implement 11 function blocks, and the performance of the function blocks according to the controller environment was measured. It was confirmed that among the 11 function blocks, the Template Matching and Classification function blocks had the highest execution time in the given integrated controller environment, whereas most other function blocks ended execution within 1 s. To examine the effect of controller performance on the execution time of different function blocks, the execution time of four function blocks was measured even in high-performance controllers. It was confirmed that the AI-based machine vision function blocks had almost no impact on CPU performance owing to their characteristics of parallel processing. Finally, performance measurement was conducted by developing an IEC 61131-3 based process automation test program capable of product classification. It was confirmed that an average of 13.8 s of execution time was required.

In future research, additional AI-based machine vision APIs that are commonly used in industrial automation systems, such as object detection and segmentation, will be developed to augment the proposed set of 11 functional blocks. Furthermore, the impact of hardware variations, such as GPU and NPU, on function blocks will be evaluated, along with research on the hardware and software architectures of controllers. Additionally, an expanded integrated development environment based on the IEC 61131-3 standard will be developed to accommodate various industrial equipment, such as PLCs, multi-axis robots, and CNC machines. Various automation programs will be developed to assess the scalability and versatility of the proposed system expansion.

**Author Contributions:** Conceptualization, S.L.; Methodology, S.L. and S.-M.H.; Software, S.-M.H.; Validation, S.L. and S.-M.H.; Formal analysis, S.L. and S.-M.H.; Resources, S.-M.H.; Data curation, S.-M.H.; Writing—original draft, U.-H.H. and S.-M.H.; Writing—review & editing, S.L. and S.-M.H.; Supervision, S.L.; Project administration, S.L.; Funding acquisition, S.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the Technology Innovation Program (No. RS-2024-00417663, 20016247) funded By the Ministry of Trade, Industry & Energy (MOTIE, Republic of Korea).

**Data Availability Statement:** The data used in this study are available on request from the corresponding author.

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

## References

1. Gsellmann, P.; Melik-Merkumians, M.; Zoitl, A.; Schitter, G. A novel approach for integrating IEC 61131-3 engineering and execution into IEC 61499. *IEEE Trans. Ind. Inform.* **2020**, *17*, 5411–5418. [CrossRef]
2. Pereira, A.; Lima, C.; Martins, J.F. The use of IEC 61131-3 to enhance PLC control and Matlab/Simulink process simulations. In Proceedings of the 2011 IEEE International Symposium on Industrial Electronics, Gdansk, Polan, 27–30 June 2011.
3. Kim, Y.; Lee, S.-Y.; Lim, S. Implementation of PLC controller connected Gazebo-ROS to support IEC 61131-3. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1.
4. Pinto, T.; Arrais, R.; Veiga, G. Bridging automation and robotics: An interprocess communication between IEC 61131-3 and ROS. In Proceedings of the 2018 IEEE 16th International Conference on Industrial Informatics (INDIN), Porto, Portugal, 18–20 July 2018.
5. Tisserant, E.; Bessard, L.; De Sousa, M. An open source IEC 61131-3 integrated development environment. In Proceedings of the 2007 5th IEEE International Conference on Industrial Informatics, Vienna, Austria, 23–27 June 2007; Volume 1.
6. Kim, I.; Kim, T.; Sung, M.; Tisserant, E.; Bessard, L.; Choi, C. An open-source development environment for industrial automation with EtherCAT and PLCopen motion control. In Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, Italy, 10–13 September 2013.
7. Papakonstantinou, N.; Sierla, S. Generating an Object Oriented IEC 61131-3 software product line architecture from SysML. In Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), Cagliari, Italy, 10–13 September 2013.
8. Kajihara, S.; Ono, M.; Houzouji, H.; Taruishi, H.; Takayanagi, Y. Development and products of the object-oriented engineering tool for the integrated controller based on IEC 61131-3. In Proceedings of the SICE 2004 Annual Conference, Sapporo, Japan, 4–6 August 2004; Volume 3.
9. Jamro, M.; Trybus, B. An approach to SysML modeling of IEC 61131-3 control software. In Proceedings of the 2013 18th International Conference on Methods & Models in Automation & Robotics (MMAR), Miedzyzdroje, Poland, 26–29 August 2013.

10. Song, S.J.; Lin, X.F.; Huang, Q.B.; Wang, C.H. An Embedded SoftLogic Control System Based on S3C44BOX and IEC 61131-3 Standard. In Proceedings of the 2007 IEEE International Conference on Control and Automation, Guangzhou, China, 30 May–1 June 2007.
11. Chodorowski, P.; Chmiel, M. IEC 61131-3 compliant PLC structure based on FPGA multi-core solution. In Proceedings of the 2016 International Conference on Signals and Electronic Systems (ICSES), Krakow, Poland, 5–7 September 2016.
12. Chmiel, M.; Kulisz, J.; Czerwinski, R.; Krzyzyk, A.; Rosol, M.; Smolarek, P. An IEC 61131-3-based PLC implemented by means of an FPGA. *Microprocess. Microsyst.* **2016**, *44*, 28–37. [CrossRef]
13. Monot, A.; Vulgarakis, A.; Behnam, M. PASA: Framework for partitioning and scheduling automation applications on multicore controllers. In Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA), Barcelona, Spain, 16–19 September 2014.
14. Vulgarakis, A.; Shooja, R.; Monot, A.; Carlson, J.; Behnam, M. Task synthesis for control applications on multicore platforms. In Proceedings of the 2014 11th International Conference on Information Technology: New Generations, Las Vegas, NV, USA, 7–9 April 2014.
15. Becker, M.; Sandström, K.; Behnam, M. A many-core based execution framework for IEC 61131-3. In Proceedings of the IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society, Yokohama, Japan, 9–12 November 2015.
16. Canedo, A.; Dalloro, L.; Ludwig, H. Pipelining for cyclic control systems. In Proceedings of the 16th international conference on Hybrid Systems: Computation and Control, Philadelphia, PA, USA, 8–11 April 2013.
17. Ardhy, F.; Hariadi, F.I. Development of SBC based machine-vision system for PCB board assembly automatic optical inspection. In Proceedings of the 2016 international symposium on electronics and smart devices (ISESD), Bandung, Indonesia, 29–30 November 2016.
18. Parakontan, T.; Sawangsri, W. Development of the machine vision system for automated inspection of printed circuit board assembl. In Proceedings of the 2019 3rd International Conference on Robotics and Automation Sciences (ICRAS), Wuhan, China, 1–3 June 2019.
19. Gerndt, R.; Michalik, S.; Krupop, S. Embedded vision system for robotics and industrial automation. In Proceedings of the 2011 9th IEEE International Conference on Industrial Informatics, Lisbon, Portugal, 26–29 July 2011.
20. Li, J.; He, M.; Su, J.; Wang, B.; Li, Z. Design and Implementation of Machine Vision Experiment Platform for Virtual Production Line. In Proceedings of the 2023 9th International Conference on Virtual Reality (ICVR), Xianyang, China, 12–14 May 2023.
21. Mihajlović, R.; Marinkov, S.; Kovačević, B.; Kaštelan, I. Challenges of Integrating Machine Vision Algorithms Based on Franca IDL into Adaptive AUTOSAR Environment. In Proceedings of the 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, 23–27 May 2022.
22. Demir, E.; Yildiz, K.; Demir, O.; Ulku, E.E. Computer Vision Based Intelligent 3D CNC Machines. In Proceedings of the 2020 Innovations in Intelligent Systems and Applications Conference (ASYU), Istanbul, Turkey, 15–17 October.
23. Krotova, N.; Pushkov, R.; Evstafieva, S. Development of a trajectory planning algorithm for moving measuring instrument for binding a basic coordinate system based on a machine vision system. In Proceedings of the 2023 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), Sochi, Russia, 15–19 May 2023.