CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF MECHANICAL ENGINEERING

Department of production machines and equipment

Bachelor’s thesis

Extension of PLC Simatic S7-1500 with machine vision system

2019 **Nikita Kuprin**

CTU
CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kuprin Nikita** Personal ID number: **467918**

Faculty / Institute: **Faculty of Mechanical Engineering**

Department / Institute: **Department of Production Machines and Equipment**

Study program: **Theoretical Fundamentals of Mechanical Engineering**

Branch of study: **No Special Fields od Study**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Extension of PLC Simatic S7-1500 with machine vision system

Bachelor’s thesis title in Czech:

Rozšíření PLC Simatic S7-1500 o systém strojového vidění

Guidelines:

Make an extension of a Siemens Simatic S7-1500 PLC training kit containing the Fischertechnik manipulator model with an existing National Instruments myRIO vision system.
Outline of work: Map communication possibilities between Siemens PLC and NI myRIO. Review the lighting options for the Fischertechnik.; Write or edit the source code for both the PLC and the vision system so that the position, color, and shape of the object in the manipulator workspace can be forwarded to the PLC via the selected communication interface.; Prepare, test and describe an example application (functional demo).; Build instructions for using the machine vision system in PLC programming classes.
Recommended range of text part: 60 - 80 pages

Bibliography / sources:

BERGER, Hans. Automating with SIMATIC S7-1500: Configuring, Programming, Motion Control and Security inside TIA Portal. Wiley VCH, 2014.

Name and workplace of bachelor’s thesis supervisor:

Ing. Lukáš Novotný, Ph.D., Department of Production Machines and Equipment, FME

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **30.04.2019** Deadline for bachelor thesis submission: **26.05.2019**

Assignment valid until: **30.09.2019**

Ing. Lukáš Novotný, Ph.D.
Supervisor’s signature

Ing. Matěj Sulitka, Ph.D.
Head of department’s signature

prof. Ing. Michael Valášek, DrSc.
Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.29.04.2019
Date of assignment receipt

Student’s signature

Declaration of Authorship

I, Nikita Kuprin, declare that this thesis titled “Extension of PLC Simatic S7-1500 with machine vision system” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have the published work of others, this is always clearly attributed
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all of the main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Date: Signature:

Acknowledgement

First and foremost, I have to thank my supervisor Ing. Novotný Lukáš, Ph.D. Without his assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished. I would like to thank you very much for your support and understanding.

I also place on record, my sense of gratitude to one and all, who directly or indirectly, have helped this thesis.

Annotation

Author: **Nikita Kuprin**

Title: **Extension of PLC Simatic S7-1500 with machine vision system**

Extent: 58 pages

Academic Year: 2018/2019

Department: Department of production machines and equipment

Tutor: Ing. Novotný Lukáš, Ph.D.

Submitter: ČVUT FS, Ú12135

Application: Build instructions for using the machine vision system in the PLC programming classes.

Keywords: machine vision, control, detection of position, LabVIEW, NI myRIO, embedded device, camera, PLC, Simatic S7-1500, OPC UA;

Abstract: The bachelor thesis deals with the making of a communication extension for the Siemens Simatic S7-1500 PLC training kit in order to transmit the position, colour, and shape from the machine vision device to the PLC. In the end, a functional pick & place robot will be created. This work is a continuation of the bachelor's thesis from Dominik Just where the machine vision device was developed [1].

Anotace

Jméno autora:	**Nikita Kuprin**
Název BP:	**Rozšíření PLC Simatic S7-1500 o systém strojového vidění**
Rozsah práce:	58 stran
Akad. rok vyhotovení:	2018/2019
Ústav:	Ústav výrobních strojů a zařízení
Vedoucí BP:	Ing. Novotný Lukáš, Ph.D.
Zadavatel tématu:	ČVUT FS, Ú12135
Využití:	Obohacení výuky ve školní laboratoři na stavebnici manipulačního robota o systém strojového vidění
Klíčová slova:	strojové vidění, řízení, detekce polohy, LabVIEW, vestavné zařízení NI myRIO, kamera, PLC, Simatic S7-1500, OPC UA;

Abstrakt:

Předložená bakalářská práce pojednává o rozšíření PLC Simatic S7-1500 o systém strojového vidění, který je aplikován na školního laboratorního robota s cílem obohatit výuku. Práce navázuje na BP Dominika Justa [1], ve které byl vyřešen vlastní systém strojového vidění ve vazbě na výukový manipulátor Fischertechnik. Navazující práce napojí strojové vidění na PLC Siemens – vyřeší komunikaci mezi strojovým viděním na platformě NI myRIO + LabVIEW a PLC Siemens, vytvoří PLC aplikaci a oživí celý systém.

Table of Contents

1 Introduction

The aim of the thesis is to make an extension of a Siemens Simatic S7-1500 PLC training kit containing the Fischertechnik manipulator model with an existing machine vision system built on National Instruments myRIO platform (shown in Figure 1). The code for both the PLC and the vision system will be edited such that the position, colour, and shape of the object in the manipulator workspace can be transmitted to the PLC via an appropriate communication interface.

Established communication will be used to create a functional pick & place demo robot for the PLC programming class. The robot should be able to recognise shape, colour of a single object in the workspace and then pick & place it accordingly. The workspace, objects and storage locations can be seen in Figure 2.

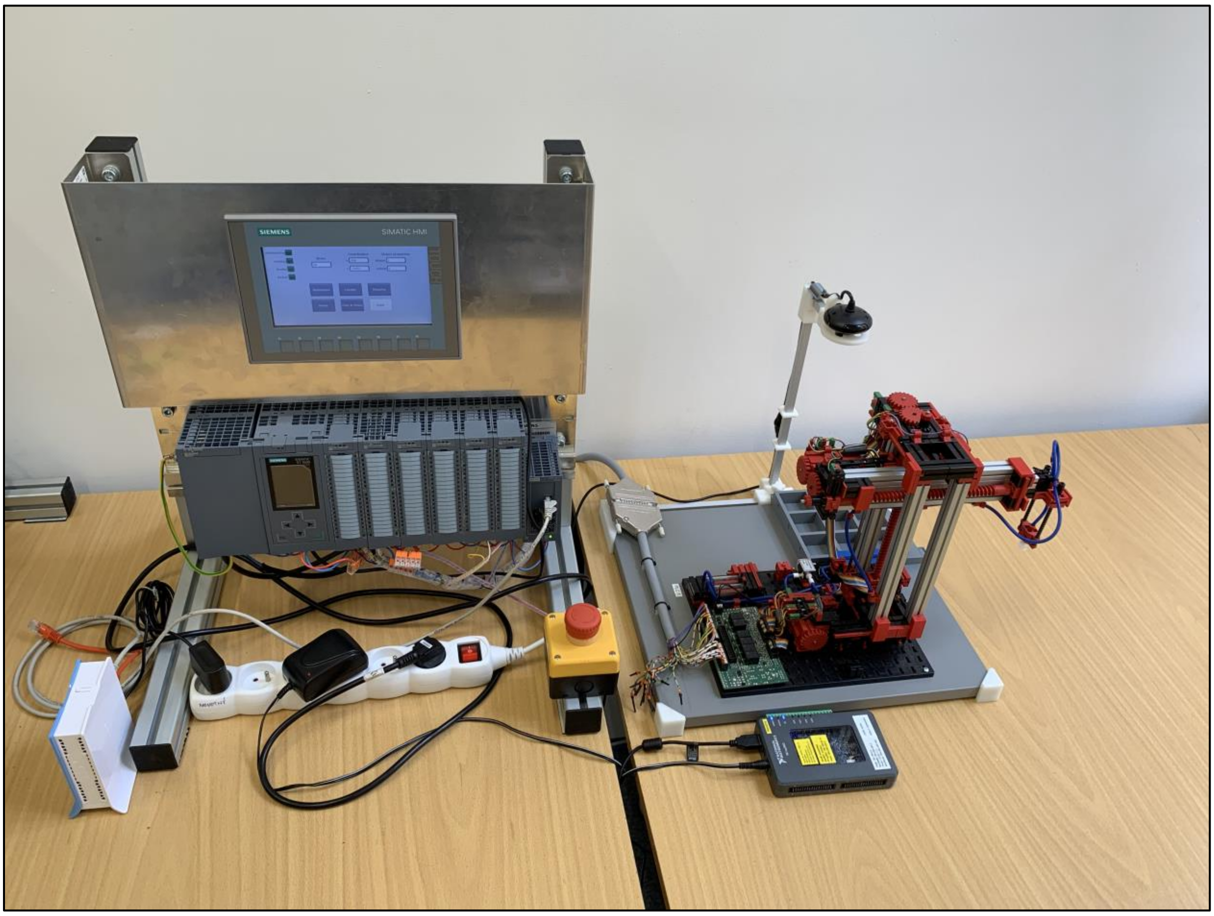


Figure 1 - PLC training kit and Fischertechnik manipulator with a vision system

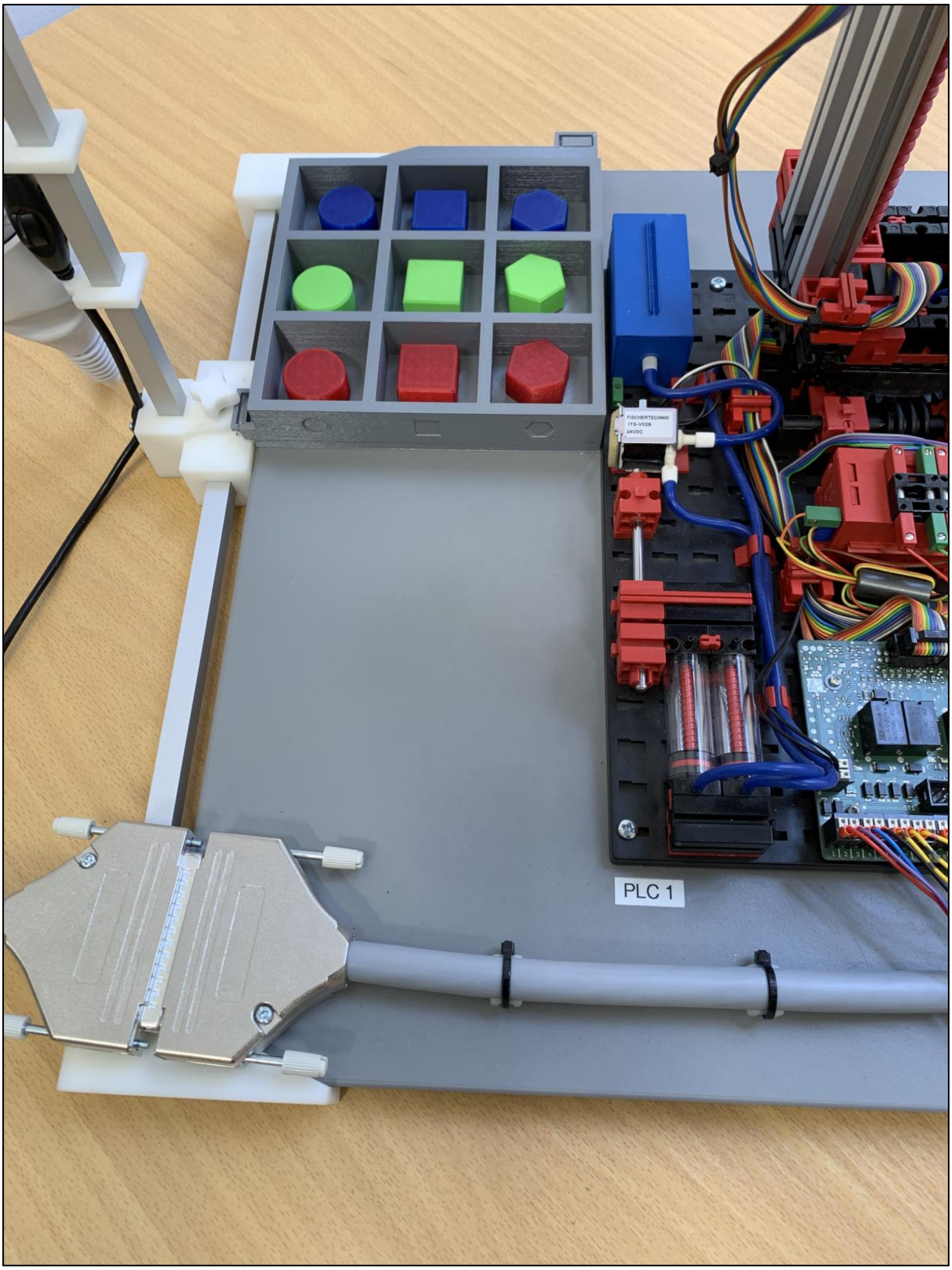


Figure 2 - Workspace and storage

2 Research

The aim of the research is to map possibilities of communication between the Siemens PLC and the NI myRIO. Also, options for improving the lighting of the Fischertechnik workspace will be reviewed.

2.1 Communication

Communication can be split into two levels: hardware and software. Hardware level is a type of network that physically interconnects the devices and software level is a communication protocol. In this project, devices of two different companies will be used: Siemens S7-1500 CPU 1516-3PN/DP and NI myRIO-1900. Hence, an open protocol must be used to allow the exchange of data between them.

2.1.1 Network

The PLC used in this project is part of the Siemens' SIMATIC S7-1500 automation system. It can be integrated across all communication standards consistently in various automation levels as demonstrated in Figure 3. [2]

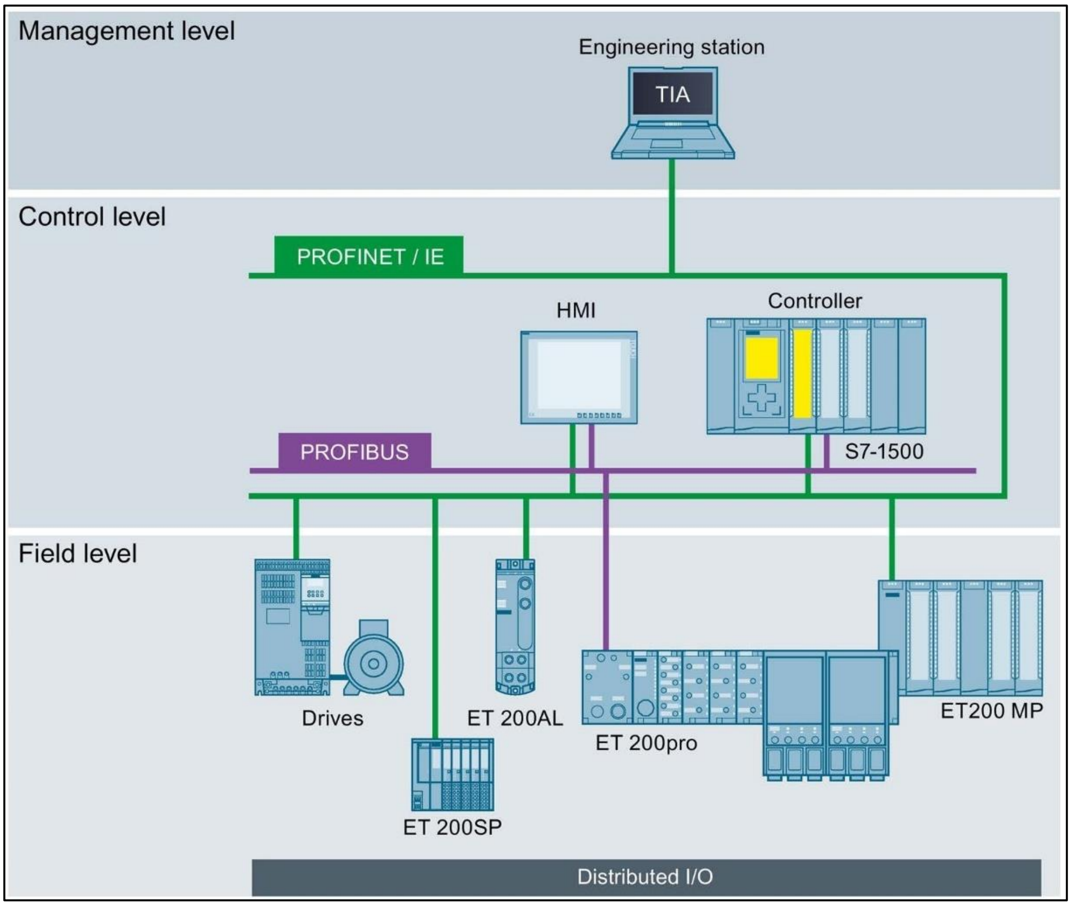


Figure 3 - SIMATIC S7-1500 at management, control and field level [2]

As can be seen from the previous image (Figure 3), two networks can be used to connect to the PLC – PROFIBUS and PROFINET/IE.

PROFIBUS is the fieldbus-based automation network. Via a single bus cable, PROFIBUS links controller or control systems with decentralized field IO devices (sensors and actuators). PROFIBUS is not only a network but also a communication protocol. It is a simple master-slave type of a protocol, where the master has full control of communication on the bus and a slave responds only when spoken to. The master records outputs and reads inputs from each of its slaves during every cycle. [3] [4]

PROFINET can be considered as a newer version of PROFIBUS designed to work with the Industrial Ethernet network. Both standards were developed by the same organization and share similarities in the engineering concepts. The biggest difference between them is that PROFINET is faster with more bandwidth and larger messages. The similarities and differences are summarized in Table 1. [3]

	PROFIBUS	**PROFINET**
organization	PI	
application profiles	same	
concepts	Engineering, GSDs	
physical layer	RS-485	Ethernet
speed	12Mbit/s	1Gbit/s or 100Mbit/s
telegram	244 bytes	1440 bytes (cyclic)^
address space	126	unlimited
technology	master/slave	provider/consumer
connectivity	PA + others*	many buses
wireless	possible*	IEEE 802.11, 15.1
motion	32 axes	>150 axes
machine-to-machine	No	Yes
vertical integration	No	Yes
^with multiple telegrams: up to 2^{32}-65 (acyclic)		
*not in spec, but solutions available		

Table 1 - PROFIBUS and PROFINET [3]

In addition to transferring data to/from IO devices, PROFINET also allows connection to the Industrial Ethernet (IE) – Ethernet network with rugged components designed specifically to work in a harsh environment [4]. Thus, in this project, the PLC can be connected to the local network via an Ethernet cable.

The local network can be created using a Wi-Fi router which would allow connecting machine vision device to the local network wirelessly (NI myRIO-1900 has a built-in Wi-Fi bus connector).

2.1.2 Communication protocol

In order to control a machine or process, signal states and numerical values are processed in the PLC. The data from the input devices is read, stored in tags, processed and sent to the output devices. [4]

In this project, the input device is NI myRIO. The data from this device cannot be read without a protocol. There is a variety of protocols supported by myRIO, out of which Modbus TCP and OPC UA are best suited for communication with PLC [5]. Those protocols are also supported by the PLC as demonstrated in Table 2 (PN/IE – PROFINET/Industrial Ethernet, DP – PROFIBUS DP).

Communication options	**PN/IE**	**DP**	**Serial**
PG communication for commissioning, testing, diagnostics	X	X	---
HMI communication for operator control and monitoring	X	X	---
Data exchange with TCP/IP, UDP, ISO-on-TCP, ISO protocol	X	---	---
As OPC UA server data exchange with OPC UA clients	X	---	---
Communication via Modbus TCP	X	---	---
Communication via UDP Multicast	X	---	---
Sending process alarms via e-mail	X	---	---
File management and file access via FTP (File Transfer Protocol); CP may be the FTP client and FTP server	X	---	---
S7 communication	X	X	---
Serial point-to-point or multi-point connection Data exchange via point-to-point with Freeport, 3964 (R), USS or Modbus protocol	---	---	X
Web server Data exchange via HTTP(S), for example for diagnostics	X	---	---
SNMP (Simple Network Management Protocol)	X	---	---
Time synchronization	X	X	---

Table 2 - Communication capabilities of the SIMATIC S7-1500 [2]

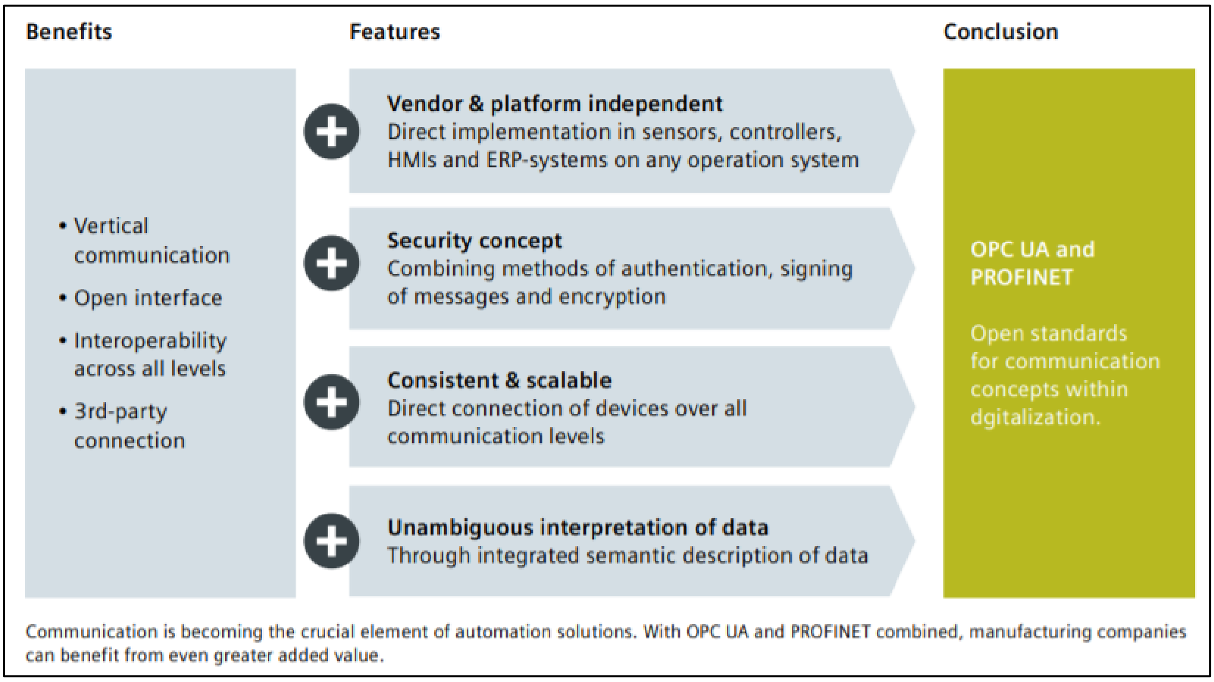
Modbus is the most established and widely used open communication protocol ideal for reliable communication of simple data. It is also a master-slave type of protocol like PROFIBUS. However, it uses the IE network instead of fieldbus-based one. The main disadvantage of the protocol is complex programming for PLC which requires the usage of special function blocks. [4] [6]

OPC UA is a client/server based communication – one or more servers wait for several clients to make requests. Once the server receives a request it answers to that and then returns into a wait state. The client controls when and what data the server will fetch from the underlying systems. Also, the client can instruct the server to send updates when such come into the server. The periodicity of the updates is controlled by the client too. [7]

UA stands for Unified Architecture. It was developed to allow multi-platform communication because originally OPC was a Windows-specific protocol. According to [8], its implementation allowed OPC to function on any of the following:

- Hardware platforms: traditional PC hardware, cloud-based servers, PLCs, micro-controllers;
- Operating Systems: Microsoft Windows, Apple OSX, Android, or any distribution of Linux.

OPC UA works smoothly with SIMATIC S7-1500. The PLC can be easily set as a server and data can be written into tags by a client directly – NI myRIO is set as a client. The key features of OPC UA can be seen in Figure 4. [9]



Benefits

- Vertical communication
- Open interface
- Interoperability across all levels
- 3rd-party connection

Features

Vendor & platform independent
Direct implementation in sensors, controllers, HMIs and ERP-systems on any operation system

Security concept
Combining methods of authentication, signing of messages and encryption

Consistent & scalable
Direct connection of devices over all communication levels

Unambiguous interpretation of data
Through integrated semantic description of data

Conclusion

OPC UA and PROFINET

Open standards for communication concepts within dgitalization.

Communication is becoming the crucial element of automation solutions. With OPC UA and PROFINET combined, manufacturing companies can benefit from even greater added value.

Figure 4 - OPC UA features [9]

2.2 Lighting

Lighting is a critical parameter for the machine vision device. It should be designed to ensure reliable functioning not dependable on external factors. The shortcoming of the current lighting solution is that it is highly dependable on surrounding illumination [1]. The goal of this section is to propose a viable solution to the issue rather than explore all the possibilities.

There are two ways in which ambient light contribution can be controlled: shield the workspace from the ambient light entirely or use a more powerful light source for illumination of the workspace [10]. Out of those, a powerful light source is a more practical option. According to the placement, three categories can be distinguished (illustrated in Figure 5):

- Backlight;
- Bright-field light;
- Dark-field light. [10]

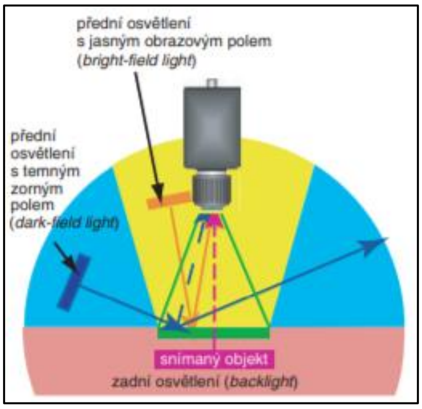


Figure 5 - Geometry of lighting [10]

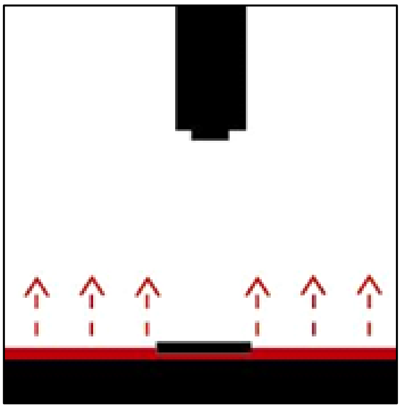


Figure 6 - Backlight [11]

2.2.1 Backlight

Backlight type of lighting illuminates the workspace from below creating a dark silhouette against a bright background as shown in Figure 6.

The backlight is commonly used for detecting the presence/absence of holes and gaps, part placing or orientating, or measuring objects [12]. Thus, this type is suitable for the required application. However, the implementation of a backlight to the existing solution requires significant modifications. Namely, it will be necessary to construct a transparent worktable under which the light source can be placed.

2.2.2 Bright-field light

Bright-field light is the most commonly used vision lighting technique. In this type, the light is reflected into the camera. Two important subtypes can be distinguished: diffuse lighting and partial bright-field lighting. Diffuse lighting (Figure 7) is usually used on shiny specular or mixed reflectivity samples where even, but multidirectional light is needed. Partial bright-field lighting (Figure 8) is directional, typically from a point source, making it a good choice for applications where good contrast and enhanced topographic details are needed. [12]

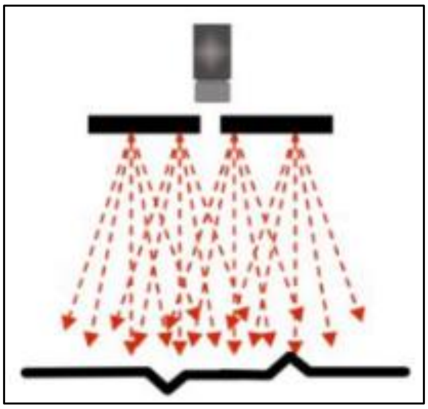


Figure 7 - Diffuse lighting [11]

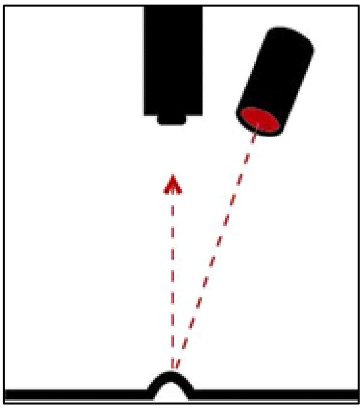


Figure 8 - Partial bright-field lighting [11]

For the required application, diffuse lighting can be easily implemented in the form of a ring light – the light source is fixed directly below the camera such that the lens passes through the centre of the light as demonstrated in Figure 9.

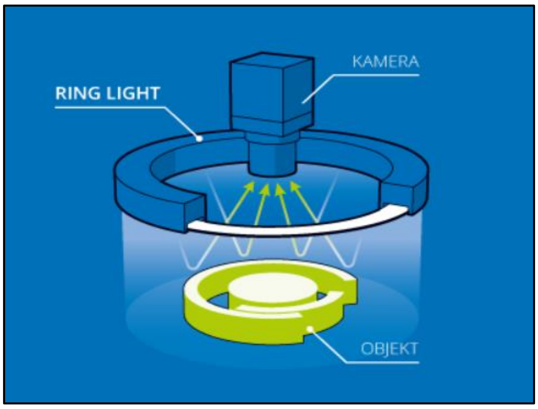


Figure 9 - Ring light [13]

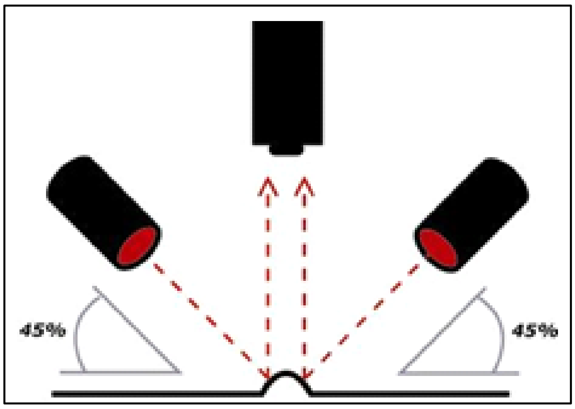


Figure 10 - Dark-field light [13]

2.2.3 Dark-field light

In this type, the light is reflected away from the camera creating a contrast image in a dark field. It is characterized by a low or medium angle of light incidence, typically requiring close proximity, illustrated in Figure 10. Dark-field light is usually used to emphasize height changes or to enhance shapes and contours.

For the given application, this type of lighting can be realized in the form of an area light source – it can be mounted on the same stand used for the camera. A commercial example can be seen in Figure 11.

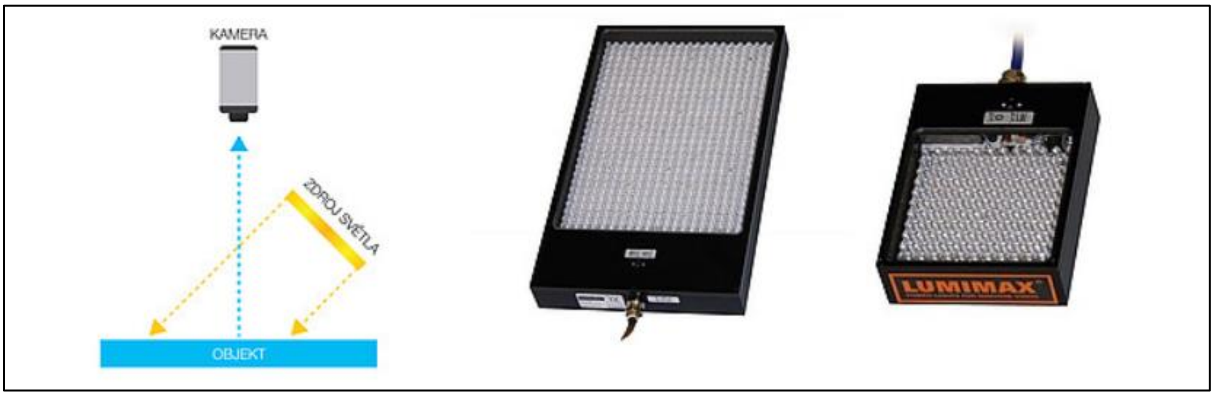


Figure 11 - Area light [11]

2.3 Research summary

For this project, the communication between the PLC (SIMATIC S7-1500) and machine vision device (NI myRIO) should be established. The purpose of communication is to forward the coordinates, shape and colour of an object in the workspace from the machine vision device to the PLC. The frequency of data transmission is controlled by the PLC – the machine vision device forwards data only when requested. According to the conducted research, it can be concluded that the best option for communication is through the Industrial Ethernet network with OPC UA protocol. This option allows the machine vision device to write the data into the PLC’s tags directly.

The most practical solution for the lighting issue is using the dark-field lighting – an area light source will be mounted on the camera stand. This solution was chosen because it does not require any modifications to the existing stand construction.

3 Solution

The solution for this project is mainly concerned with writing the source codes. The hardware is suitable in the original state except for the lighting. The setup can be seen in Figure 12.

For the connection of devices on the hardware level, the local network was created using the router. The PLC is connected to the local network through the PROFINET interface using the Ethernet cable and the NI myRIO is connected wirelessly through the Wi-Fi. For administration, a PC or a laptop can be also connected to the local network through either Ethernet cable or Wi-Fi. (Figure 13)

The solution will be realized in the following way: the OPC UA communication principles of the demo programs will be implemented into the robot control and the inspection programs, the codes for both PLC and LabView will be optimized for pick & place operation, a new light source will be added.

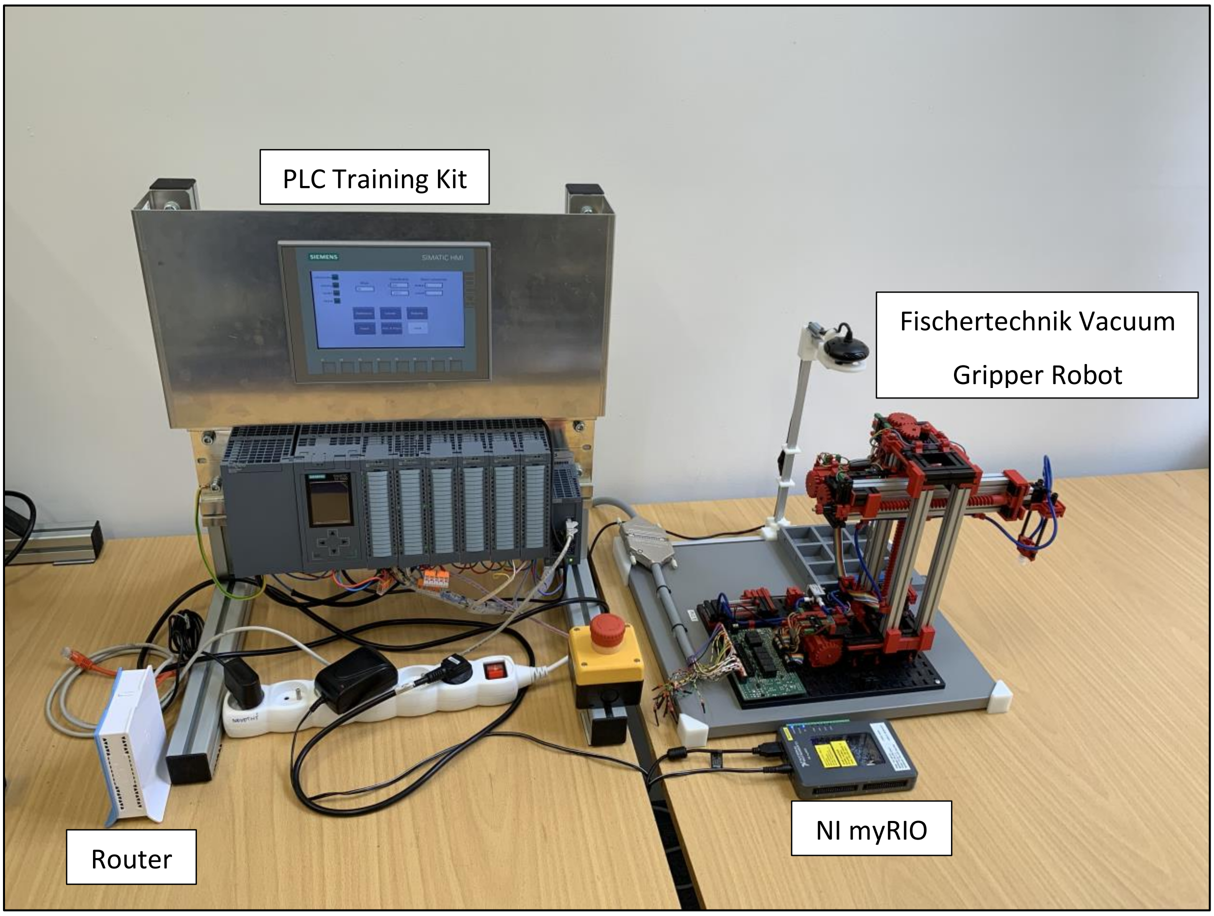


Figure 12 - Hardware setup

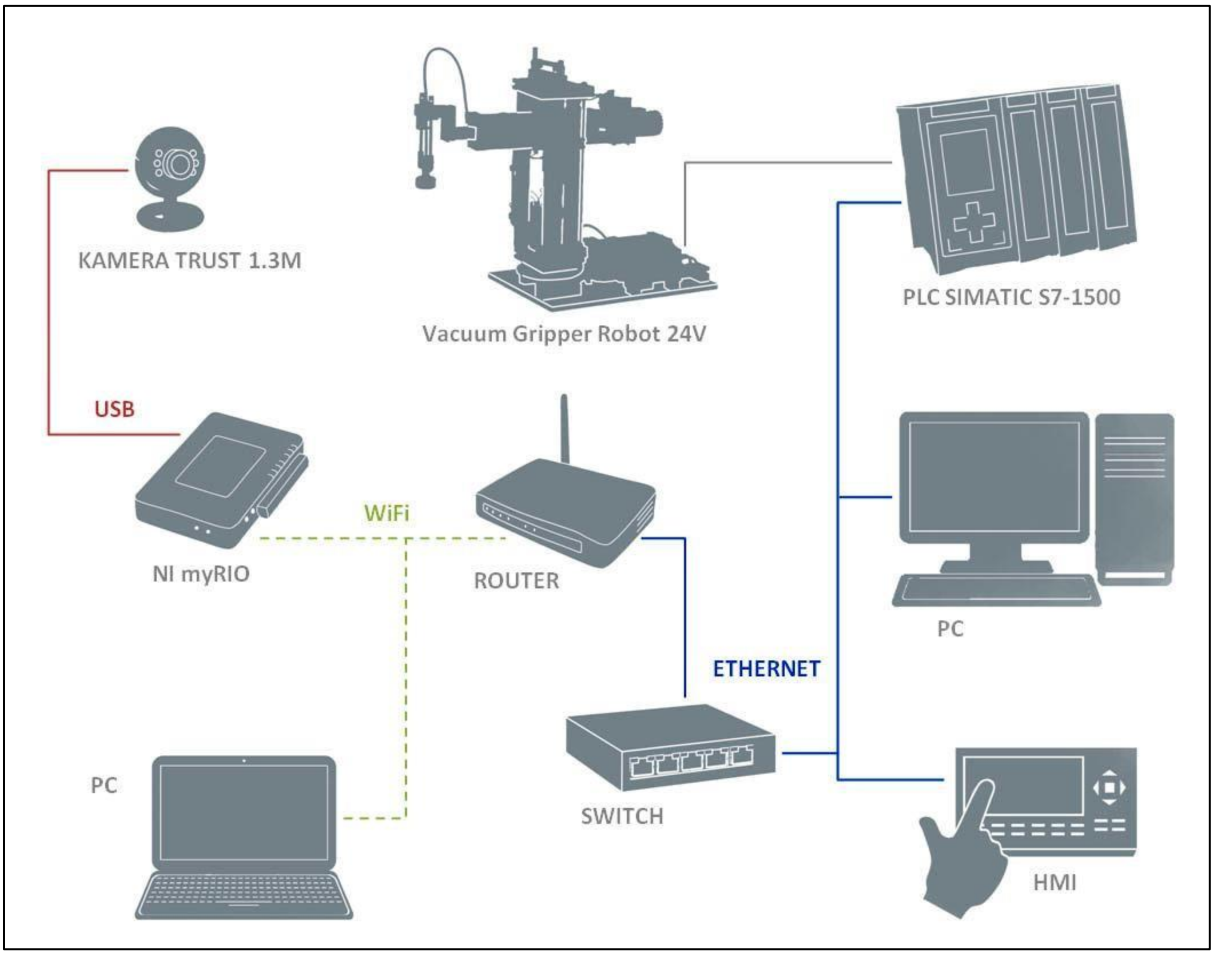


Figure 13 - Connection scheme [1]

Four programs will be used as a basis for the solution (all the codes can be found on the CD in the folder "Original programs"):

1. Robot control ("PLC-Robot_V15") – TIA Portal program for the Vacuum Gripper Robot control;
2. Image inspection ("program s komentáři") – LabView program for the image analysis;
3. LabView OPC UA demo ("OPC UA komunikace") – LabVIEW program with OPC UA principles;
4. TIA Portal OPC UA demo ("PLC_komunikace_OPC_UA_v1_V14") – TIA Portal program with OPC UA principles.

The initial state of the programs is described in the next section. The last program ("PLC_komunikace_OPC_UA_v1_V14") is omitted from the description because it only serves as a demonstration for setting up the OPC UA server. Those settings are explained in detail in the manual (see Attachment 3a).

3.1 Initial state of the programs

3.1.1 Robot control

The program was created for the PLC programming course. It was written in the Siemens TIA Portal software using the SCL language. Initially, the program was designed to have the following functionality: "Jog" – navigate the manipulator to the desired location without an ability to relocate objects; "Manual" – navigate the manipulator to the predetermined locations with an ability to relocate objects; "Automatic" – automatic relocation of objects between the conveyors.

The code is organized in blocks of 3 types: Organization Blocks, Function Blocks and Data Blocks as illustrated in Figure 14.

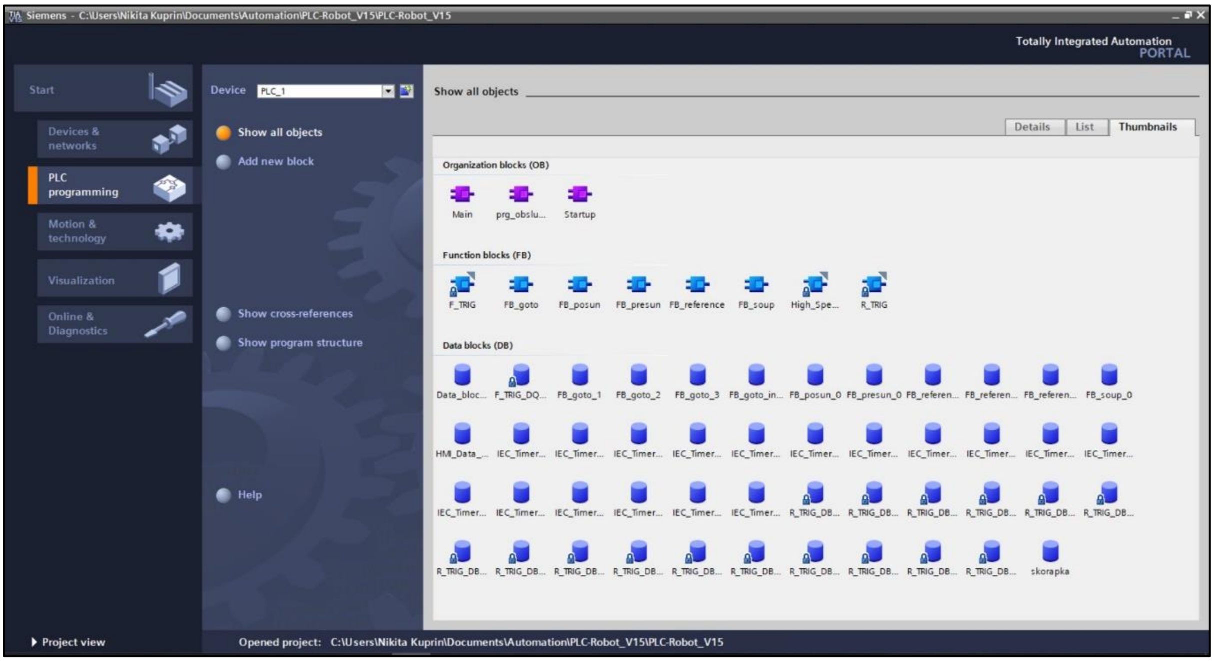


Figure 14 - Blocks

Organization blocks (OBs) form the interface between the operating system and the user program. They are called by the operating system and control the following operations:

- Start-up characteristics of the automation system;
- Cyclic program processing;
- Interrupt-driven program execution;
- Error handling. [4]

Functions (FCs) are code blocks without memory. There is no data memory in which values of block parameters can be stored. Therefore, when a function is called, all formal parameters must be assigned actual parameters. Functions can use global data blocks to store data permanently. [4]

Function blocks (FBs) are code blocks that store their (FCs) input, output and in-out parameters permanently in instance **Data blocks** (DBs), so that they remain available even after the block has been executed. Therefore, they are also referred to as blocks "with memory". Function blocks can also operate with temporary tags. Temporary tags are not stored in the instance DB but are available for one cycle only. [4]

Data blocks are used to store program data. Thus, DBs contain variable data that is used by the user program. Global data blocks store data that can be used by all other blocks. [4]

Important blocks from the actual program that will be used in this project are described in Table 3.

Name	Type	Description
Main [OB1]	OB	The block for cyclic program processing – the most important part of the code which controls everything. It is built on the principle of a state machine. (see Attachment 1a)
Startup [OB100]	OB	The block where the values of tags are initialized. This block is run only one time when the PLC is turned on. (see Attachment 1b)
prg_obsluha_DO [OB123]	OB	The OB for servicing the digital outputs from the HMI panel. (see Attachment 1c)
FB_reference [FB1]	FB	FB for referencing the position – manipulator goes to the default position and then counters are reset to zero. The default position is recognized by the switches at the end of each axis. (see Attachment 1d)
FB_posun [FB4]	FB	The FB is responsible for guiding the manipulator to the desired location with the help another FB – FB_goto. (see Attachment 1d).
FB_presun [FB3]	FB	The FB for picking up an object and relocation it to a designated location. It also uses FB_goto. (see Attachment 1e)
FB_goto [FB2]	FB	Controls the motors of the manipulator such that they go to the required position. (see Attachment 1f)

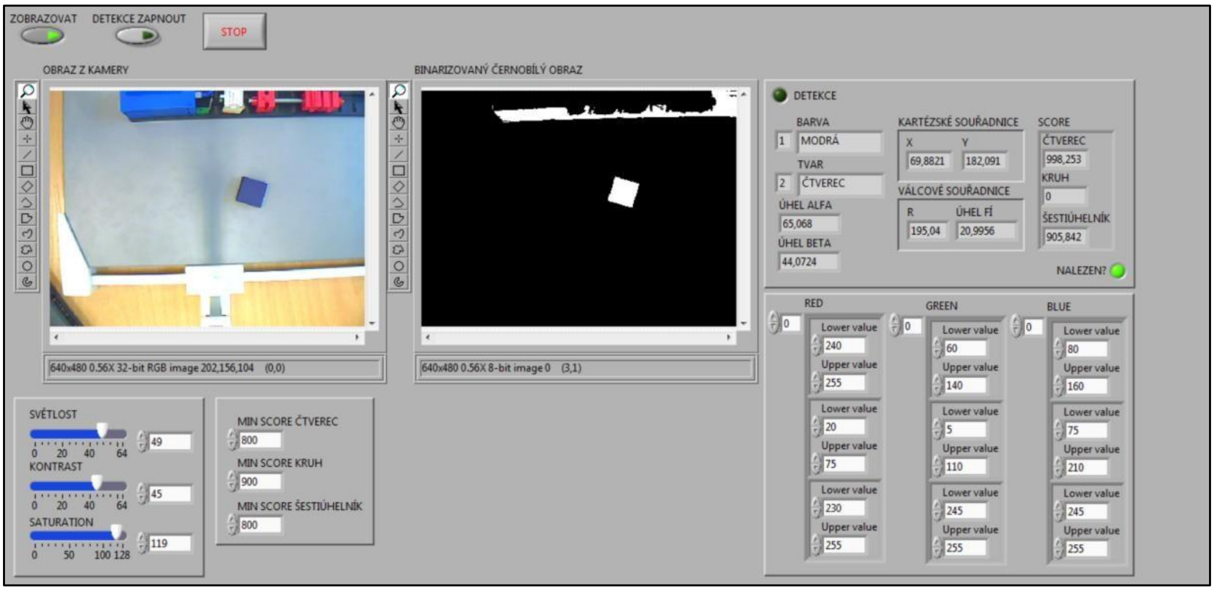
Table 3 - Blocks that will be used in the project

3.1.2 Image inspection

This program was developed by Dominik Just as a part of his bachelor's thesis [1]. It is written using a visual programming language of LabView software to run on the NI myRIO embedded system. The following functionality was initially present in the program:

- Detection of shape, colour, position and orientation of the object in the workspace;
- Writing of the output data to the network such that it can be accessed through a dedicated NI "Data Dashboard" application;
- Ability to start and control the application by a hardware button on the myRIO device.

The code itself is written in the block diagram (Figure 16) and the front panel is used for the user interface (Figure 15). The working principle of the program is explained in details in [1].



640x480 0.56X 32-bit RGB image 202,156,104 (0,0)

640x480 0.56X 8-bit image 0 (3,1)

DETEKCE

BARVA: 1 MODRÁ

TVAR: 2 ČTVEREC

ÚHEL ALFA: 65,068

ÚHEL BETA: 44,0724

KARTÉZSKÉ SOUŘADNICE: X 69,8821, Y 182,091

VÁLCOVÉ SOUŘADNICE: R 195,04, ÚHEL FÍ 20,9956

SCORE: ČTVEREC 998,253; KRUH 0; ŠESTIÚHELNÍK 905,842

NALEZEN? ●

RED	GREEN	BLUE
0	0	0
Lower value 240	Lower value 60	Lower value 80
Upper value 255	Upper value 140	Upper value 160
Lower value 20	Lower value 5	Lower value 75
Upper value 75	Upper value 110	Upper value 210
Lower value 230	Lower value 245	Lower value 245
Upper value 255	Upper value 255	Upper value 255

SVĚTLOST 49

KONTRAST 45

SATURATION 119

MIN SCORE ČTVEREC 800

MIN SCORE KRUH 900

MIN SCORE ŠESTIÚHELNÍK 800

Figure 15 - Front panel of the Inspection program [1]

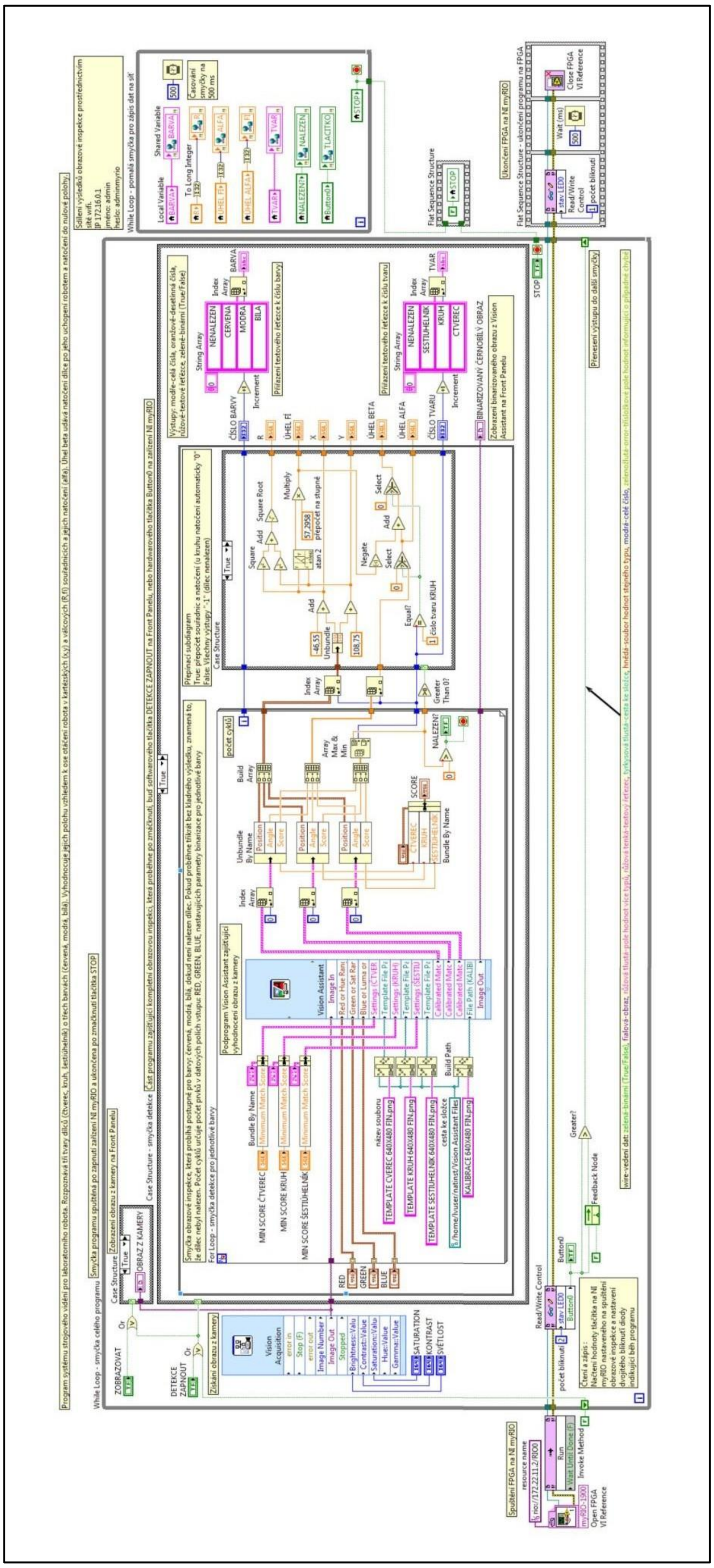
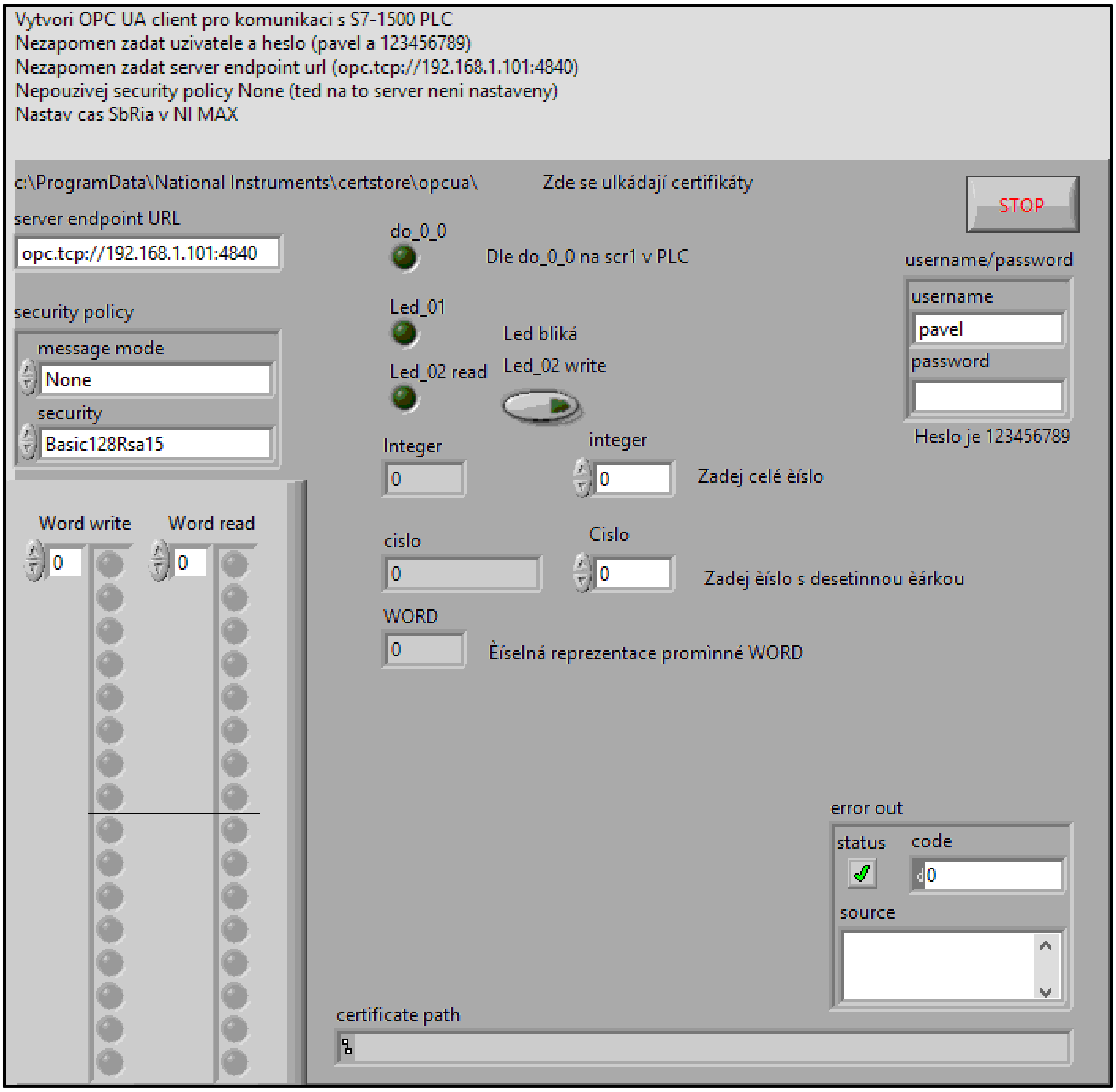


Figure 16 - Block diagram of the Inspection program [1]

3.1.3 LabVIEW OPC UA Demo

This program will be used as a guide for implementing OPC UA communication. It shows how to connect a LabView application to the server as a client. The front panel can be seen in Figure 17 and the block diagram can be seen in Figure 18, Figure 19.



Vytvori OPC UA client pro komunikaci s S7-1500 PLC
Nezapomen zadat uzivatele a heslo (pavel a 123456789)
Nezapomen zadat server endpoint url (opc.tcp://192.168.1.101:4840)
Nepouzivej security policy None (ted na to server neni nastaveny)
Nastav cas SbRia v NI MAX

c:\ProgramData\National Instruments\certstore\opcua\ Zde se ulkádají certifikáty

server endpoint URL: opc.tcp://192.168.1.101:4840

security policy
message mode: None
security: Basic128Rsa15

Word write: 0
Word read: 0

do_0_0 Dle do_0_0 na scr1 v PLC

Led_01 Led bliká

Led_02 read

Led_02 write

Integer: 0

integer: 0 Zadej celé èíslo

cislo: 0

Cislo: 0 Zadej èíslo s desetinnou èárkou

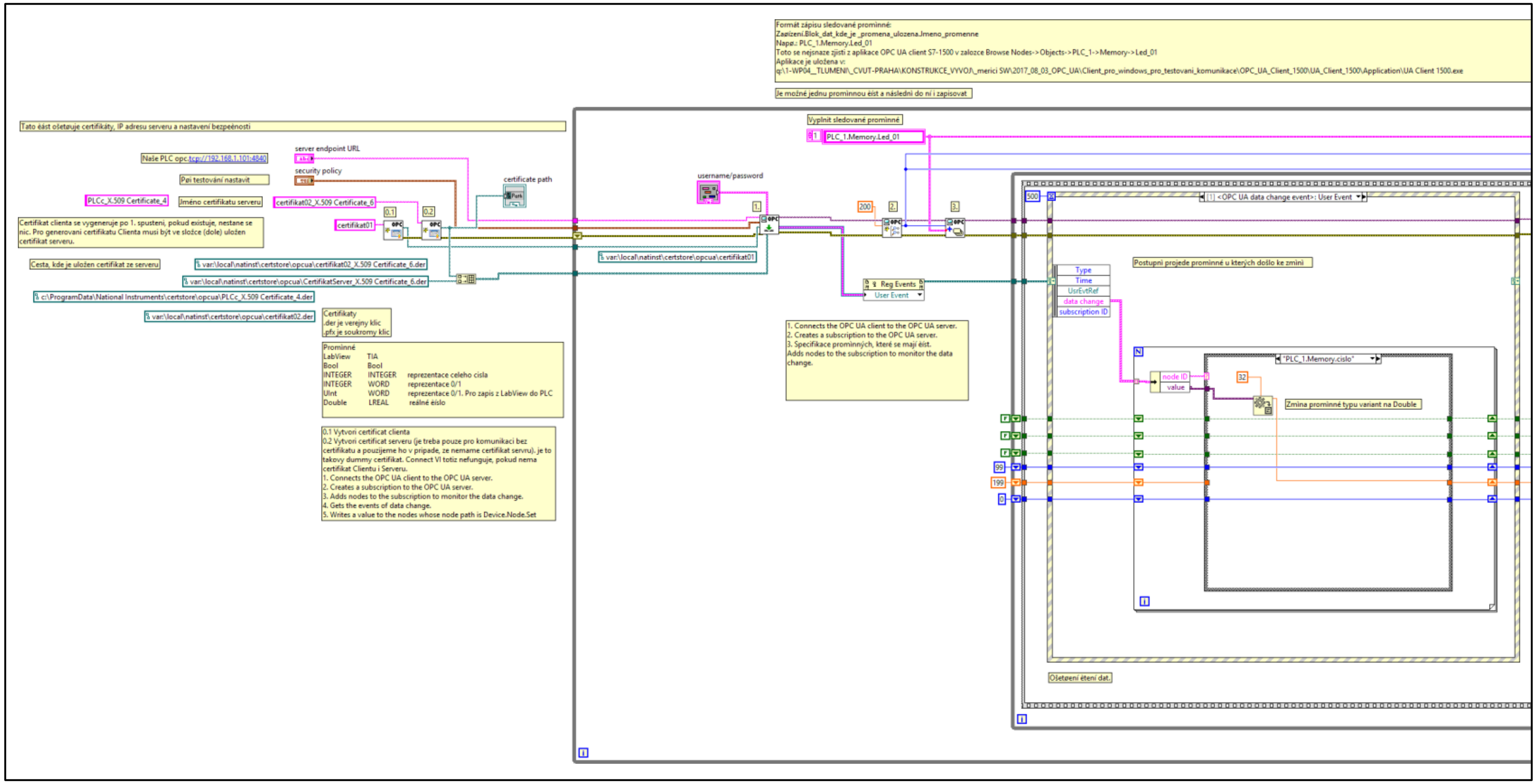
WORD: 0 Èíselná reprezentace promìnné WORD

username/password
username: pavel
password:
Heslo je 123456789

error out
status: ✓
code: 0
source:

certificate path

Figure 17 - Front panel of the Demo program



Formát zápisu sledované promìnné:
Zaøízení.Blok_dat_kde_je _promena_ulozena.Jmeno_promenne
Napø.: PLC_1.Memory.Led_01
Toto se nejsnaze zjisti z aplikace OPC UA client S7-1500 v zalozce Browse Nodes->Objects->PLC_1->Memory->Led_01
Aplikace je uložena v:
q:\1-WP04__TLUMENI_CVUT-PRAHA\KONSTRUKCE_VYVOJ_merici SW\2017_08_03_OPC_UA\Client_pro_windows_pro_testovani_komunikace\OPC_UA_Client_1500\UA_Client_1500\Application\UA Client 1500.exe

Je možné jednu promìnnou èíst a následnì do ní i zapisovat

Tato èást ošetøuje certifikáty, IP adresu serveru a nastavení bezpeènosti

Naše PLC opc.tcp://192.168.1.101:4840

Pøi testování nastavit

Jméno certifikatu serveru

Certifikat clienta se vygeneruje po 1. spusteni, pokud existuje, nestane se nic. Pro generovani certifikatu Clienta musi být ve složce (dole) uložen certifikat serveru.

Cesta, kde je uložen certifikat ze serveru

Certifikaty
.der je verejny klic
.pfx je soukromy klic

Promìnné

LabView	TIA	
Bool	Bool	
INTEGER	INTEGER	reprezentace celeho cisla
INTEGER	WORD	reprezentace 0/1
UInt	WORD	reprezentace 0/1. Pro zapis z LabView do PLC
Double	LREAL	reálné èíslo

0.1 Vytvori certificat clienta
0.2 Vytvori certificat serveru (je treba pouze pro komunikaci bez certifikatu a pouzijeme ho v pripade, ze nemame certifikat servru). je to takovy dummy certifikat. Connect VI totiz nefunguje, pokud nema certifikat Clientu i Serveru.
1. Connects the OPC UA client to the OPC UA server.
2. Creates a subscription to the OPC UA server.
3. Adds nodes to the subscription to monitor the data change.
4. Gets the events of data change.
5. Writes a value to the nodes whose node path is Device.Node.Set

Vyplnit sledované promìnné

1. Connects the OPC UA client to the OPC UA server.
2. Creates a subscription to the OPC UA server.
3. Specifikace promìnných, které se mají èíst.
Adds nodes to the subscription to monitor the data change.

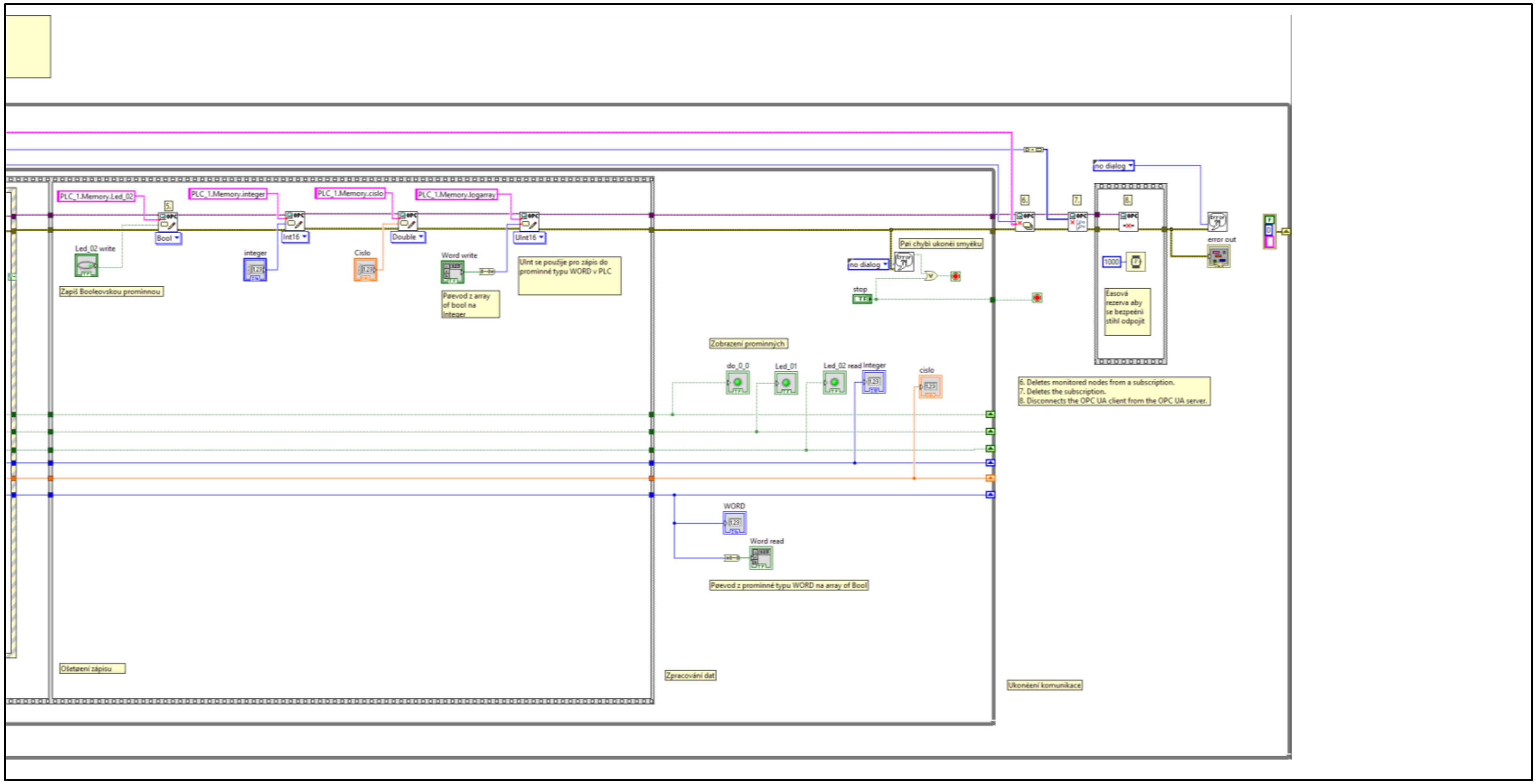
Postupnì projede promìnné u kterých došlo ke zmìnì

Zmìna promìnné typu variant na Double

Ošetøení ètení dat.

server endpoint URL
security policy
certificate path
PLCc_X.509 Certificate_4
certifikat02_X.509 Certificate_6
certifikat01
0.1
0.2
var:\local\natinst\certstore\opcua\certifikat02_X.509 Certificate_6.der
var:\local\natinst\certstore\opcua\CertifikatServer_X.509 Certificate_6.der
c:\ProgramData\National Instruments\certstore\opcua\PLCc_X.509 Certificate_4.der
var:\local\natinst\certstore\opcua\certifikat02.der
username/password
var:\local\natinst\certstore\opcua\certifikat01
PLC_1.Memory.Led_01
1.
2.
3.
200
Reg Events
User Event
500
[1] <OPC UA data change event>: User Event
Type
Time
UsrEvtRef
data change
subscription ID
N
node ID
value
32
"PLC_1.Memory.cislo"
99
199
0

Figure 18 - Block diagram of the Demo program (part 1 – left)



Zapiš Booleovskou prominnou

Pøevod z array of bool na Integer

UInt se použije pro zápis do prominné typu WORD v PLC

Pøi chybì ukonèi smyèku

Zobrazení promìnných

Pøevod z promìnné typu WORD na array of Bool

Èasová rezerva aby se bezpeènì stihl odpojit

6. Deletes monitored nodes from a subscription.
7. Deletes the subscription.
8. Disconnects the OPC UA client from the OPC UA server.

Ošetøení zápisu

Zpracování dat

Ukonèení komunikace

Figure 19 - Block diagram of the Demo program (part 2 – right)

3.2 Modification of the programs

3.2.1 Robot control

Modification of the PLC program is mostly concerned with the implementation of the pick & place sequence (pick & place is later referred to as p&p). The communication part is simple and only requires changing settings as described in Attachment 3a.

The goal of the program is to navigate the manipulator to the object, pick it up and place it to the designated location. Properties of the objects are analysed by the inspection program. Objects will be stored in a 3D printed box with slots for each colour/shape combination (Figure 20). Each slot's location coordinates are stored in tags *pos_x_ik* and *pos_c_ik*. The tags will be explained later in Table 4.

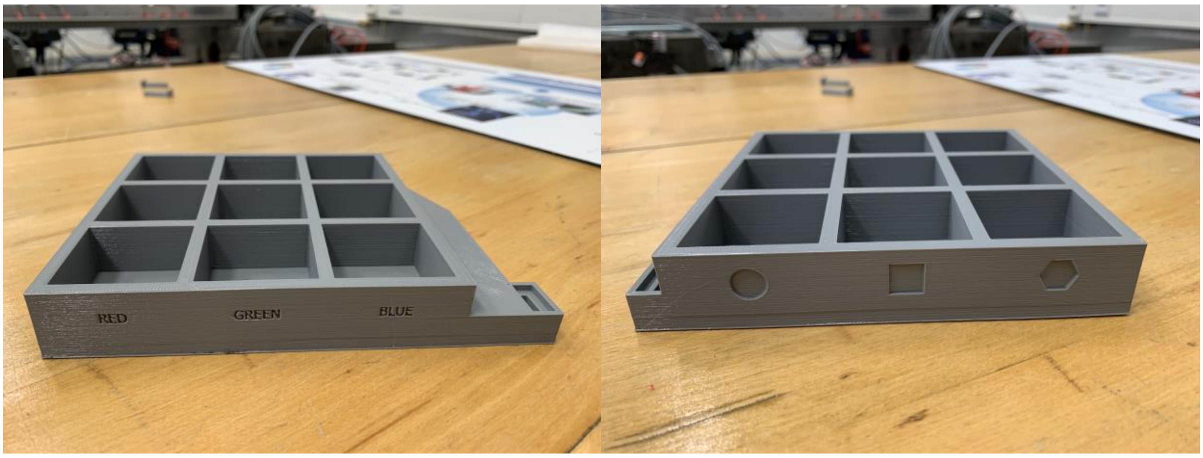


Figure 20 - Storage

For calibration purposes, there also should be a fully manual mode with an ability to move manipulator in any axis and read actual position from high speed counter (in increments). The operation of vacuum gripper will be included in the manual mode as well.

The following modifications will be made in order to achieve the goal:

1. Addition of extra tags;
2. Addition of manual mode with HMI screen;
3. Addition of P&P mode with HMI screen;
4. Addition of storage coordinates to the Startup OB;
5. Restructure of the Main OB;
6. Clean up of unnecessary modes.

3.2.1.1 Extra tags

It is necessary to add extra tags to exchange the data between the PLC and inspection program. Also, the coordinates of the storage locations and manual mode require their tags. All the new tags, except the ones for the manual mode, can be seen in Table 4:

Name	Data type	I/O	Function
locate, LocateExec	Bool	Output	Starting the image analysis sequence in the inspection program. The tag is operated by the button on the HMI panel – when the button is pressed tag's value changes to 1 and the sequence is initiated.
found	Bool	Input	Indicating whether the inspection program found an object.
Shape	LReal	Input	Numerical equivalent to the shape of an object.
Colour	LReal	Input	Numerical equivalent to the colour of an object.
pos_x	LReal	Input	Position of an object in the polar coordinate system – radius. The value is in increments.
pos_c	LReal	Input	Position of an object in the polar coordinate system – angle. The value is in increments.
pos_x/c_ik	LReal	(PLC's local data)	Coordinates of storage slots for each colour (i) and shape (k) combination: 0 – red, 1 – blue, 2 – green; 0 – hexagon, 1 – square, 2 – circle (for example 00 is red hexagon).

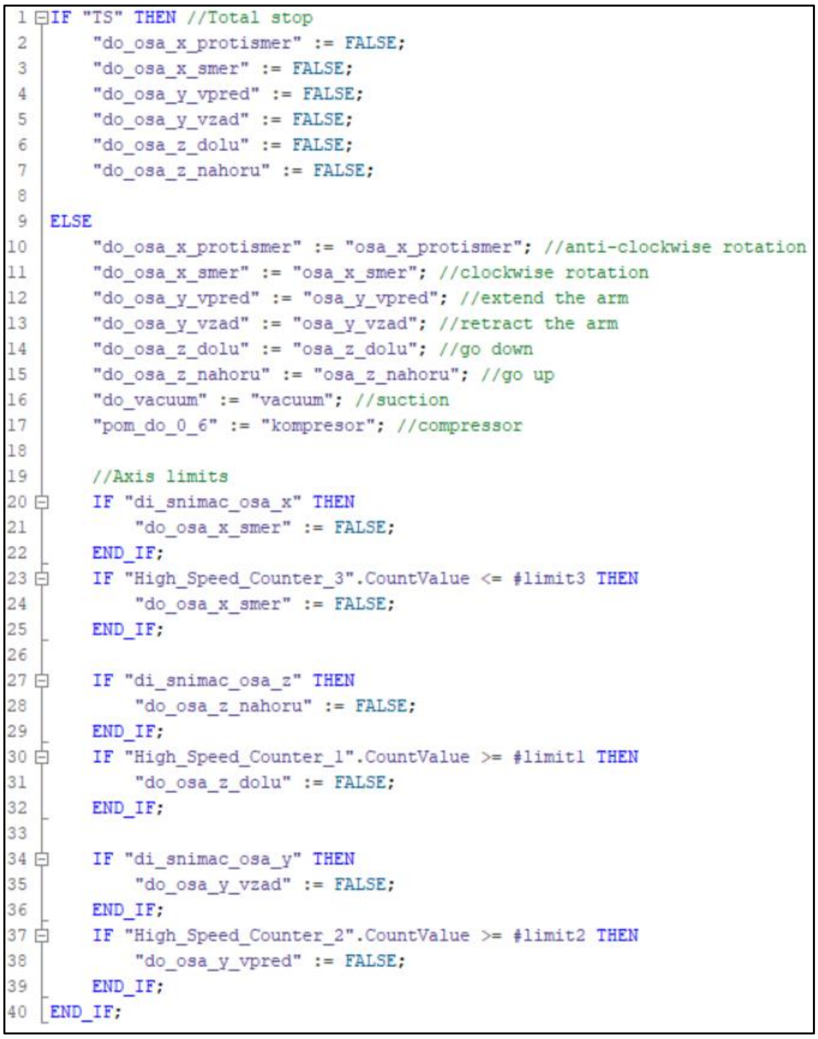
Table 4 - New tags

3.2.1.2 Manual mode

As already mentioned, the purpose of the manual mode is to allow free movement of the manipulator in any axis. This will be realized by the addition of a new function block *Manual*, which will be called from the Main OB.

Manual function block allows output tags to be controlled by the tags assigned to the HMI buttons. For the safe operation, Total Stop and axis limits were added – if Total Stop has been pressed or axis limits have been reached the motors stop immediately. The code for this block can be seen in Figure 21.

HMI screen has buttons to control axis motors, turn on/off compressor and suction, perform referencing, return to the root screen, resume after total stop. It also has screens indicating the state of the Main OB and readings from the high speed counters. The layout of the screen can be seen in Figure 22.

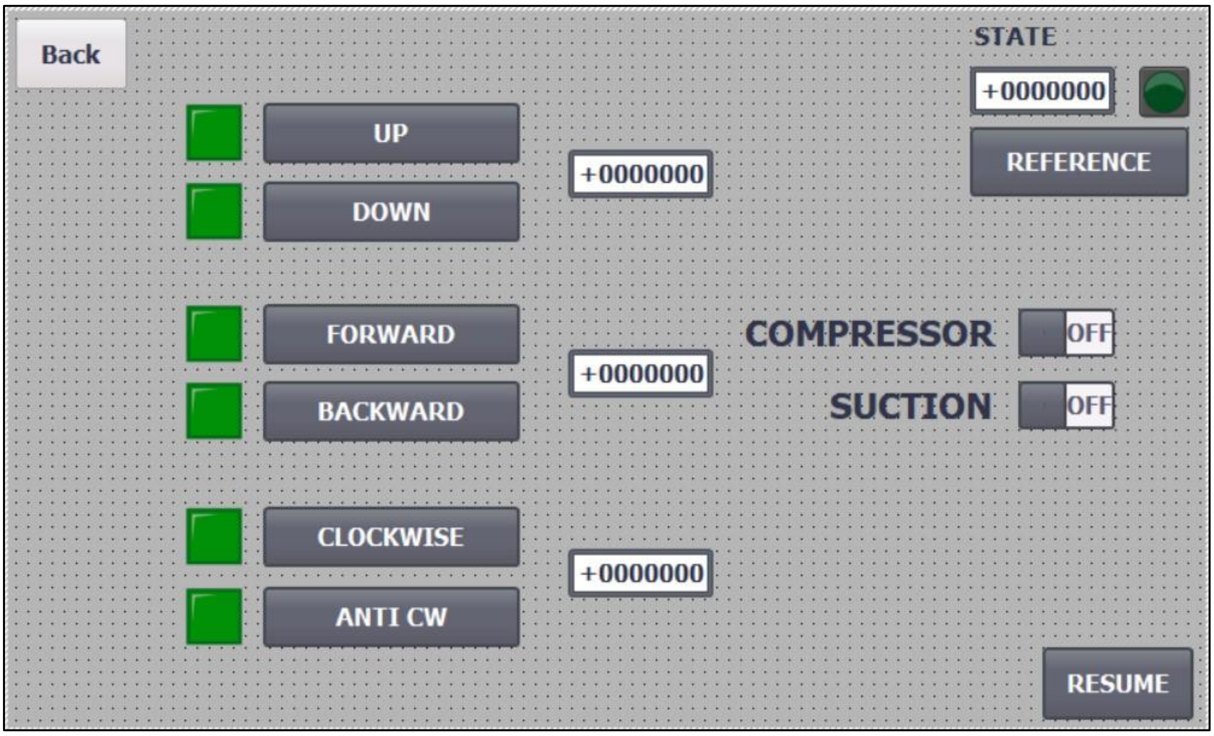


```
IF "TS" THEN //Total stop
    "do_osa_x_protismer" := FALSE;
    "do_osa_x_smer" := FALSE;
    "do_osa_y_vpred" := FALSE;
    "do_osa_y_vzad" := FALSE;
    "do_osa_z_dolu" := FALSE;
    "do_osa_z_nahoru" := FALSE;

ELSE
    "do_osa_x_protismer" := "osa_x_protismer"; //anti-clockwise rotation
    "do_osa_x_smer" := "osa_x_smer"; //clockwise rotation
    "do_osa_y_vpred" := "osa_y_vpred"; //extend the arm
    "do_osa_y_vzad" := "osa_y_vzad"; //retract the arm
    "do_osa_z_dolu" := "osa_z_dolu"; //go down
    "do_osa_z_nahoru" := "osa_z_nahoru"; //go up
    "do_vacuum" := "vacuum"; //suction
    "pom_do_0_6" := "kompresor"; //compressor

    //Axis limits
    IF "di_snimac_osa_x" THEN
        "do_osa_x_smer" := FALSE;
    END_IF;
    IF "High_Speed_Counter_3".CountValue <= #limit3 THEN
        "do_osa_x_smer" := FALSE;
    END_IF;

    IF "di_snimac_osa_z" THEN
        "do_osa_z_nahoru" := FALSE;
    END_IF;
    IF "High_Speed_Counter_1".CountValue >= #limit1 THEN
        "do_osa_z_dolu" := FALSE;
    END_IF;

    IF "di_snimac_osa_y" THEN
        "do_osa_y_vzad" := FALSE;
    END_IF;
    IF "High_Speed_Counter_2".CountValue >= #limit2 THEN
        "do_osa_y_vpred" := FALSE;
    END_IF;
END_IF;
```

Figure 21 - Code for the Manual FB

3.2.1.3 P&P mode

There is no dedicated function block for P&P mode. It is rather integrated into the Main OB as a state. The code will be explained in the next section. HMI screen has buttons to reference, reset, locate, pick & place, resume after Total Stop and go back to the root screen. The wide variety of indicators allows to see the state, object location and properties, undergoing operation (for example referencing). The layout can be seen in Figure 23.



STATE +0000000

+0000000

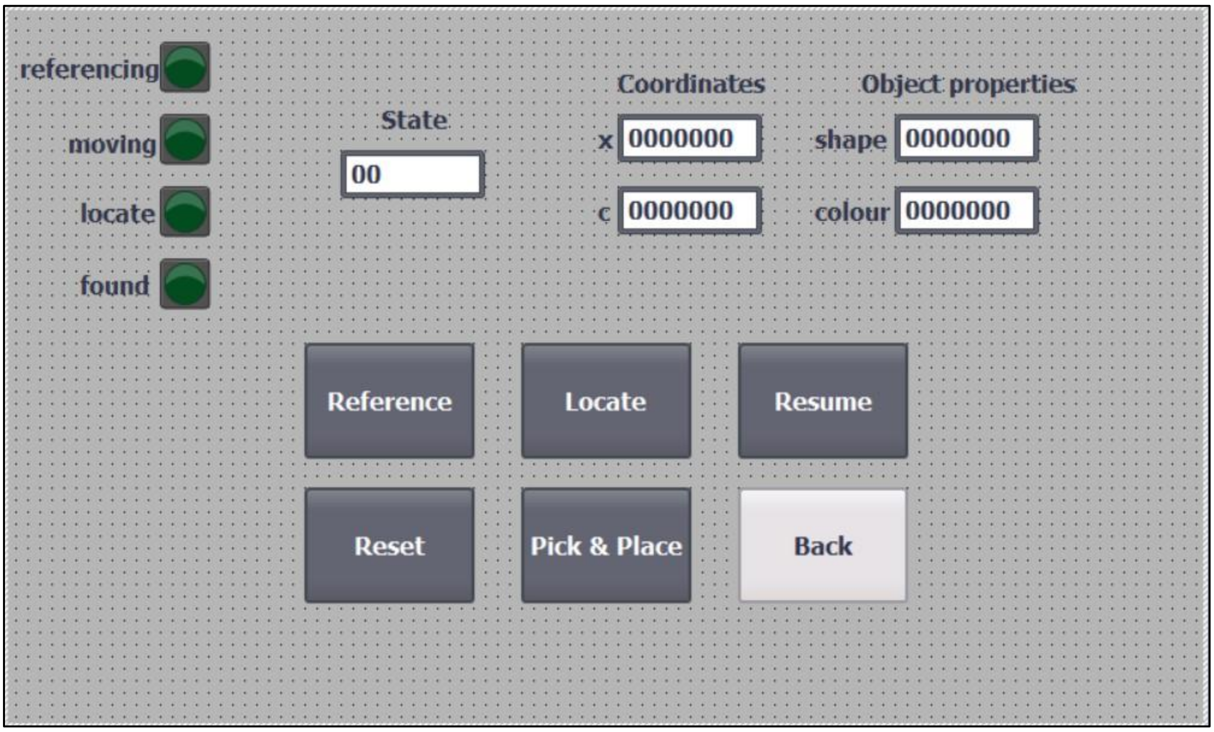
+0000000

+0000000

COMPRESSOR OFF

SUCTION OFF

Figure 22 - HMI screen for manual mode



State 00

Coordinates

x 0000000

c 0000000

Object properties

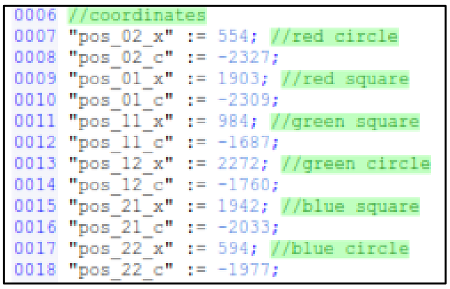
shape 0000000

colour 0000000

Figure 23 - HMI screen for P&P mode

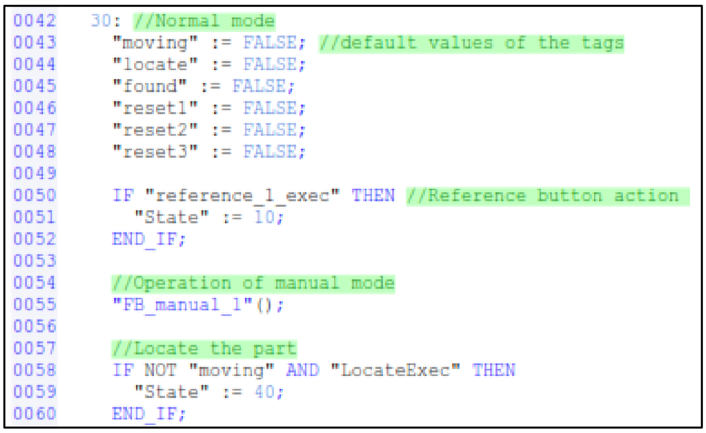
3.2.1.4 Startup OB

The only modification of the Startup OB is an addition of storage slots coordinates. They were obtained using the manual mode – manipulator's suction cup was positioned above a slot and readings from the encoders were recorded. Their values (in increments) can be seen in Figure 24 (full code can be found in the Attachment 1i).



```
//coordinates
"pos_02_x" := 554; //red circle
"pos_02_c" := -2327;
"pos_01_x" := 1903; //red square
"pos_01_c" := -2309;
"pos_11_x" := 984; //green square
"pos_11_c" := -1687;
"pos_12_x" := 2272; //green circle
"pos_12_c" := -1760;
"pos_21_x" := 1942; //blue square
"pos_21_c" := -2033;
"pos_22_x" := 594; //blue circle
"pos_22_c" := -1977;
```

Figure 24 - Coordinates of storage locations



```
  30: //Normal mode
    "moving" := FALSE; //default values of the tags
    "locate" := FALSE;
    "found" := FALSE;
    "reset1" := FALSE;
    "reset2" := FALSE;
    "reset3" := FALSE;

    IF "reference_1_exec" THEN //Reference button action
      "State" := 10;
    END_IF;

    //Operation of manual mode
    "FB_manual_1"();

    //Locate the part
    IF NOT "moving" AND "LocateExec" THEN
      "State" := 40;
    END_IF;
```

Figure 25 - Normal mode

3.2.1.5 Main OB

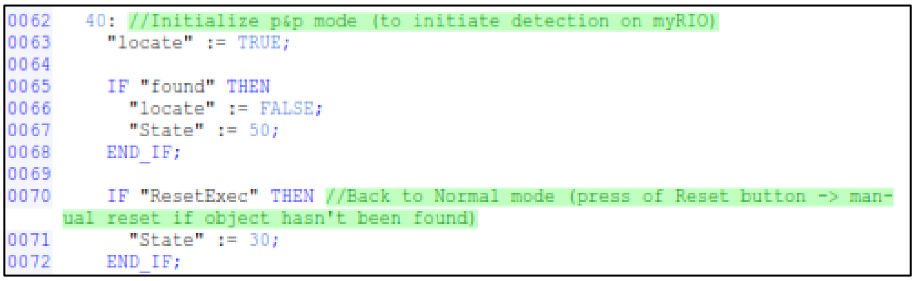
Main OB is the most crucial block – cyclic program processing happens there. The cases in the state machine were modified. There are 6 states in total: 0, 10, 20, 30, 40, 50. States 0-20 and the code outside the case structure are kept original. The other states are described below (full code can be found in the Attachment 1g).

State 30 – normal mode (Figure 25):

- Default values of *locate* and *found* tags are initialized – both are false;
- FB *manual* is called – allows operation of manipulator manually;
- Condition for switching to the next state – press of the *Locate* button on the HMI screen sets the tag *LocateExec* to 1 and the program goes to the state 40.
- Alternative method for exiting the state – press of the *Reference* button returns the program to state 10.

State 40 – Initialize p&p mode (Figure 26):

- The image analysis sequence of the inspection program is initiated;
- Condition for switching to the next state – the tag *found* is set to 1 when the object has been located;
- Alternative method of exiting the state – press of the *Reset* button on the HMI screen sets the tag *ResetExec* to 1 and returns the program to the normal mode.



```
 40: //Initialize p&p mode (to initiate detection on myRIO)
   "locate" := TRUE;

   IF "found" THEN
     "locate" := FALSE;
     "State" := 50;
   END_IF;

   IF "ResetExec" THEN //Back to Normal mode (press of Reset button -> manual reset if object hasn't been found)
     "State" := 30;
   END_IF;
```

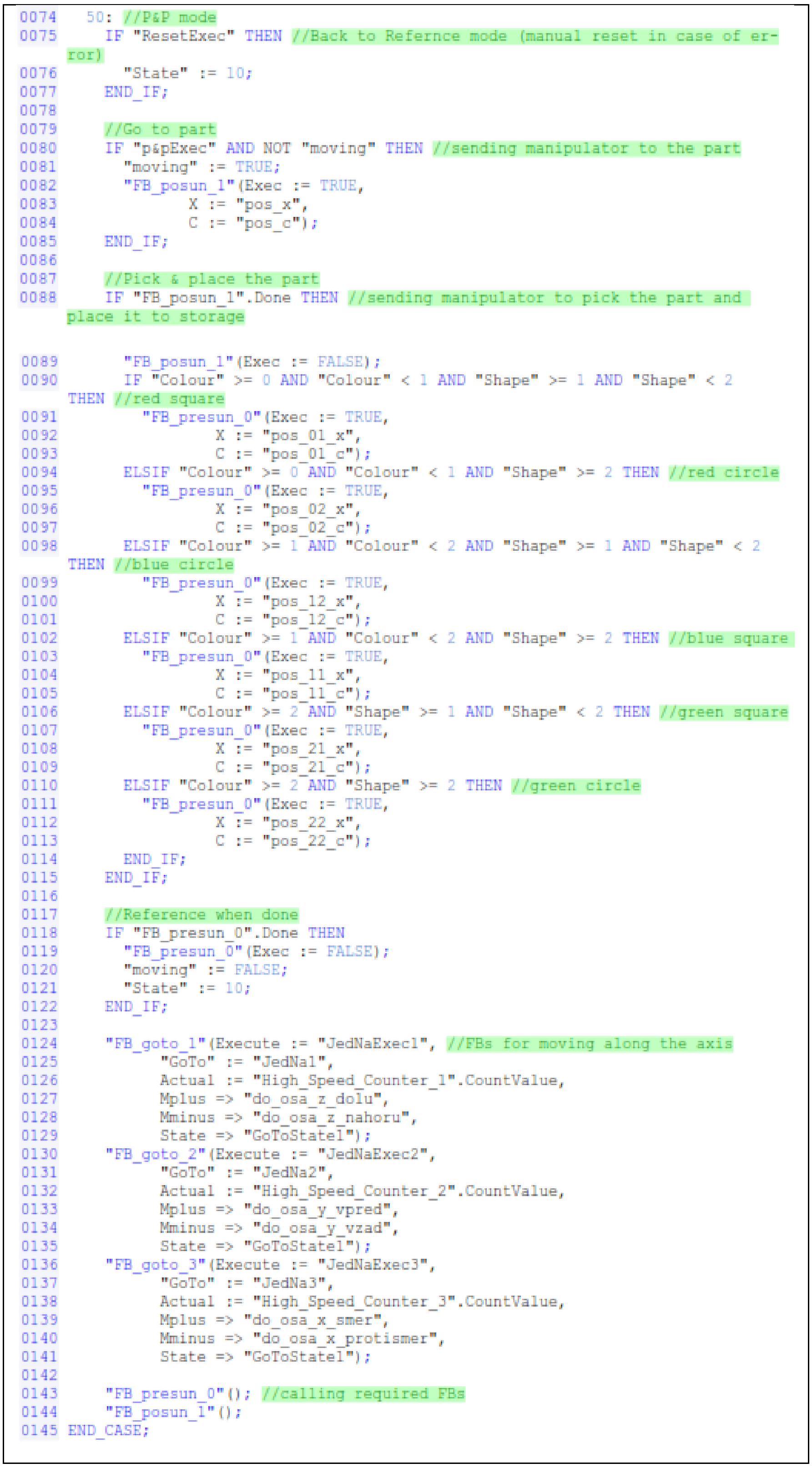
Figure 26 - Initialize p&p mode

State 50 – P&P mode (Figure 27):

- The manipulator is navigated to the object with the use of the function block *FB_Posun* – position of the object is written to the tags *pos_x* and *pos_c*;
- The object is placed to the designated location by the function block *FB_Presun* depending on the colour/shape combination;
- Function block *FB_goto* is called to initiate motors for movement along the required axis – *FB_goto_1* for vertical (z-axis), *FB_goto_2* for horizontal (x-axis), *FB_goto_3* for rotation (c-axis);
- Condition for switching to the next state – after the object has been placed to the required location, state changes to 10;
- Alternative method for exiting the state – press of the *Reset* or the *Reference* button returns the program to the appropriate state.

3.2.1.6 Other modifications

The codes of other blocks were left intact, only comments in English were added and an extra line of code was added to *FB_posun* (see Attachment 1j). Also, unnecessary modes, such as “*Automatic”, “Jog”, “Skorapky”* were deleted from the Main OB. Their HMI screens were removed as well.



```
  50: //P&P mode
    IF "ResetExec" THEN //Back to Refernce mode (manual reset in case of er-
ror)
      "State" := 10;
    END_IF;

    //Go to part
    IF "p&pExec" AND NOT "moving" THEN //sending manipulator to the part
      "moving" := TRUE;
      "FB_posun_1"(Exec := TRUE,
             X := "pos_x",
             C := "pos_c");
    END_IF;

    //Pick & place the part
    IF "FB_posun_1".Done THEN //sending manipulator to pick the part and
place it to storage

      "FB_posun_1"(Exec := FALSE);
      IF "Colour" >= 0 AND "Colour" < 1 AND "Shape" >= 1 AND "Shape" < 2
THEN //red square
        "FB_presun_0"(Exec := TRUE,
                X := "pos_01_x",
                C := "pos_01_c");
      ELSIF "Colour" >= 0 AND "Colour" < 1 AND "Shape" >= 2 THEN //red circle
        "FB_presun_0"(Exec := TRUE,
                X := "pos_02_x",
                C := "pos_02_c");
      ELSIF "Colour" >= 1 AND "Colour" < 2 AND "Shape" >= 1 AND "Shape" < 2
THEN //blue circle
        "FB_presun_0"(Exec := TRUE,
                X := "pos_12_x",
                C := "pos_12_c");
      ELSIF "Colour" >= 1 AND "Colour" < 2 AND "Shape" >= 2 THEN //blue square
        "FB_presun_0"(Exec := TRUE,
                X := "pos_11_x",
                C := "pos_11_c");
      ELSIF "Colour" >= 2 AND "Shape" >= 1 AND "Shape" < 2 THEN //green square
        "FB_presun_0"(Exec := TRUE,
                X := "pos_21_x",
                C := "pos_21_c");
      ELSIF "Colour" >= 2 AND "Shape" >= 2 THEN //green circle
        "FB_presun_0"(Exec := TRUE,
                X := "pos_22_x",
                C := "pos_22_c");
      END_IF;
    END_IF;

    //Reference when done
    IF "FB_presun_0".Done THEN
      "FB_presun_0"(Exec := FALSE);
      "moving" := FALSE;
      "State" := 10;
    END_IF;

    "FB_goto_1"(Execute := "JedNaExec1", //FBs for moving along the axis
          "GoTo" := "JedNa1",
          Actual := "High_Speed_Counter_1".CountValue,
          Mplus => "do_osa_z_dolu",
          Mminus => "do_osa_z_nahoru",
          State => "GoToState1");
    "FB_goto_2"(Execute := "JedNaExec2",
          "GoTo" := "JedNa2",
          Actual := "High_Speed_Counter_2".CountValue,
          Mplus => "do_osa_y_vpred",
          Mminus => "do_osa_y_vzad",
          State => "GoToState1");
    "FB_goto_3"(Execute := "JedNaExec3",
          "GoTo" := "JedNa3",
          Actual := "High_Speed_Counter_3".CountValue,
          Mplus => "do_osa_x_smer",
          Mminus => "do_osa_x_protismer",
          State => "GoToState1");

    "FB_presun_0"(); //calling required FBs
    "FB_posun_1"();
END_CASE;
```

Figure 27 - P&P mode

3.2.2 Inspection program

This program is responsible for detecting the object in the workspace, analysing its properties and forwarding the data to the PLC via OPC UA communication protocol. Image analysis functionality is already present in the original program. Thus, it is necessary to implement the OPC UA communication and to add the conversion from distance/angle to increments.

The structure of the program needs to be modified as well. The main program is placed in the **main.vi** and contains two loops – one for communication and the other for image inspection. To keep the program tidy, it was decided to optimize the loops such that each one would fit entirely into a PC screen. Hence, the code was split into smaller sections known as subVIs (a subVI is similar to a subroutine in text-based programming languages). The advantage of this technique is that it allows using a small icon instead of bulky code in the block diagram as demonstrated in Figure 28 and Figure 29. [14]

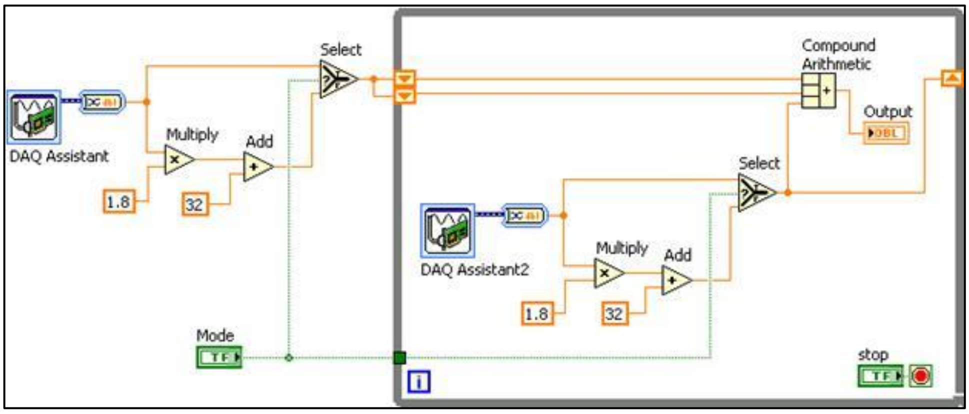


Figure 28 - Code before using a subVI [14]

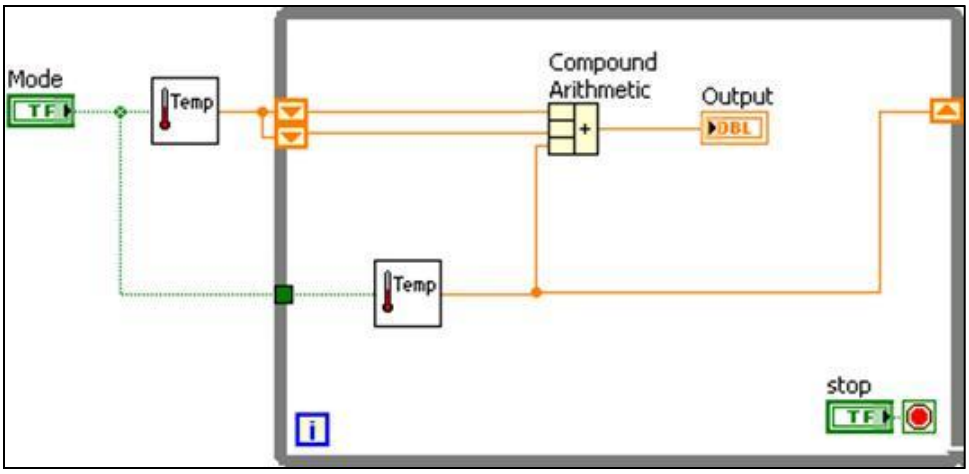


Figure 29 - Code after using a subVI [14]

The required modifications are summarized below.

1. Block diagram (see Attachment 2a):
   a. Addition of OPC UA communication in the form of an additional while loop;
   b. Addition of math operations for conversion from distance/angle to increments;
   c. Optimization.
2. Front panel (see Attachment 2b):
   a. Translation to English;
   b. Addition of an extra tab for OPC UA communication.

3.2.2.1 OPC UA communication

As already mentioned above, the inspection program will be connected to the OPC UA server (PLC) as a client. LabView has an extensive function library for OPC UA communication that can be found in **Functions → Data Communication → OPC UA → OPC UA Client** (Figure 30). It contains various VIs (VI is a LabVIEW code file), that will be used to establish communication.

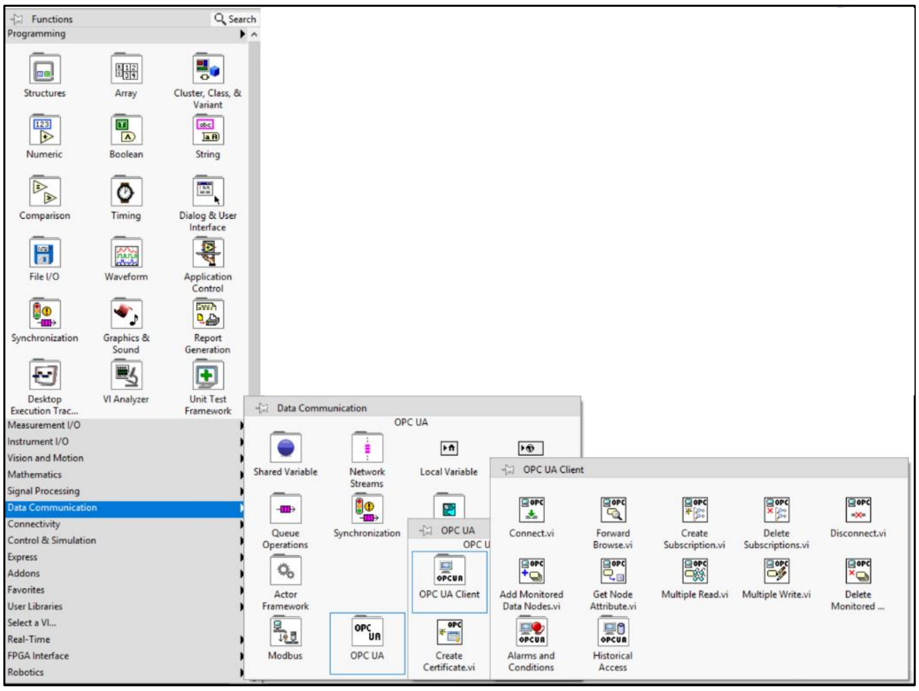


Figure 30 - OPC UA VIs

The communication is implemented into the inspection program as an additional **While Loop** (Figure 50). Firstly, it is necessary to create server and client certificates. This needs to be done because **Connect.vi** (used in the next step) does not work without them. For simplification, connection to the server (PLC) is realized without security. Hence, the created certificates are dummy (they do not provide any security).

Certificates are created using **Create Certificate.vi**. Their paths are added to an array which is then used by **Connect.vi**. The operation is performed only one time during startup. Also, indicators are added such that paths can be seen in the front panel. The principle of how certificates are created can be seen in Figure 31.

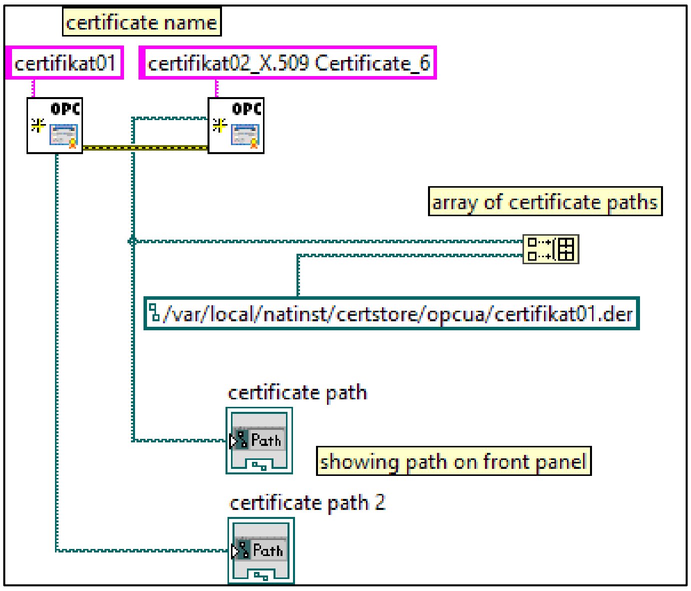

certificate name
certifikat01
certifikat02_X.509 Certificate_6
OPC
OPC
array of certificate paths
/var/local/natinst/certstore/opcua/certifikat01.der
certificate path
Path
showing path on front panel
certificate path 2
Path

Figure 31 - Creating certificates

Next, it is necessary to connect to the OPC UA server. This will be done using the **Connect.vi**. Several terminals must be wired to the VI as demonstrated in Figure 32. The realization in the actual program can be seen in Figure 36.

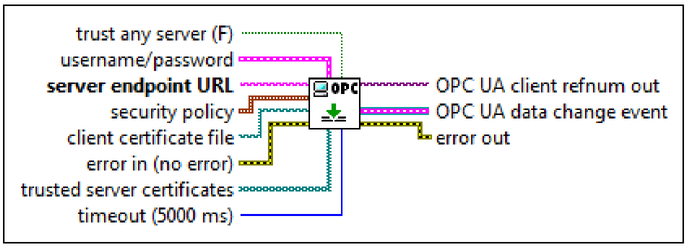
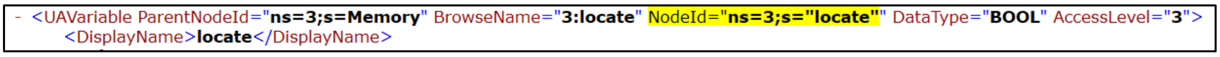


Figure 32 - "Connect" VI wiring

After the connection to the server has been established, it is necessary to subscribe to the nodes (PLC tags) of the server using **Create Subscription.vi**. This operation is done to incur data changes on the server in order to monitor when tags change values (for example to register when detection is enabled on the HMI). Overall, 5 tags are monitored: *locate, Calibrate, Red_cn, Blue_cn, Green_cn* (last 3 tags are used for calibration and will be explained later). To monitor these tags specifically, it is necessary to add their node IDs to the subscription. This is done using **Add Monitored Nodes.vi**. The IDs of the nodes can be found in the TIA Portal by going to the PLC's settings (as described in the Attachment 3a) **OPC UA → Export → Export OPC UA XML file**. For example, a node ID for the *locate* tag is *ns=3;s="locate"* as shown in Figure 33. The IDs in precisely the same format are then entered to an array and wired to the VI.



```
- <UAVariable ParentNodeId="ns=3;s=Memory" BrowseName="3:locate" NodeId="ns=3;s="locate"" DataType="BOOL" AccessLevel="3">
      <DisplayName>locate</DisplayName>
```

Figure 33 - Node ID example

There is no need to repeat the operations described above for the whole duration of the program. Thus, they can be put outside of the communication loop in a **Flat Sequence Structure** to ensure that the loop starts only after the communication has been established (Figure 35). Also, this is exactly part of the code that can be put into a subVI as was explained at the beginning of the chapter. The subVI will be called **connecting to the OPC UA server**. Its block diagram is shown in Figure 36 and the icon with wiring can be seen in Figure 34.

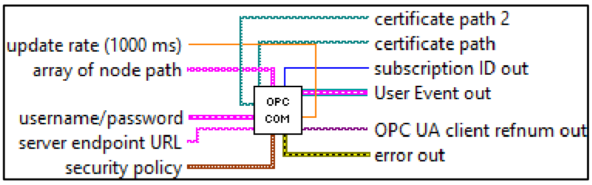


Figure 34 - Icon for "connecting to the OPC UA server" subVI

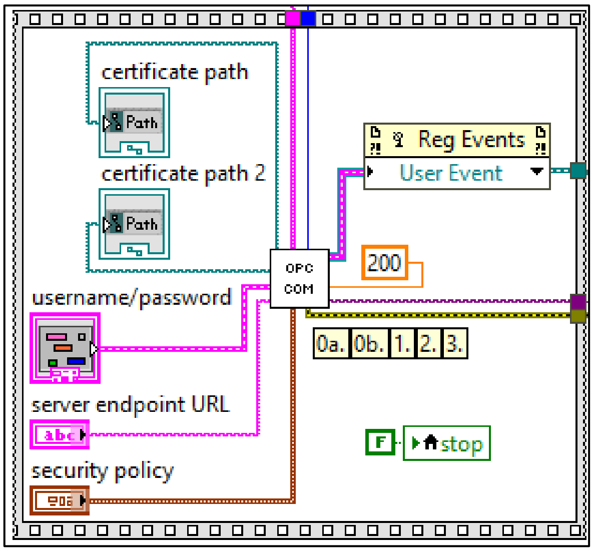
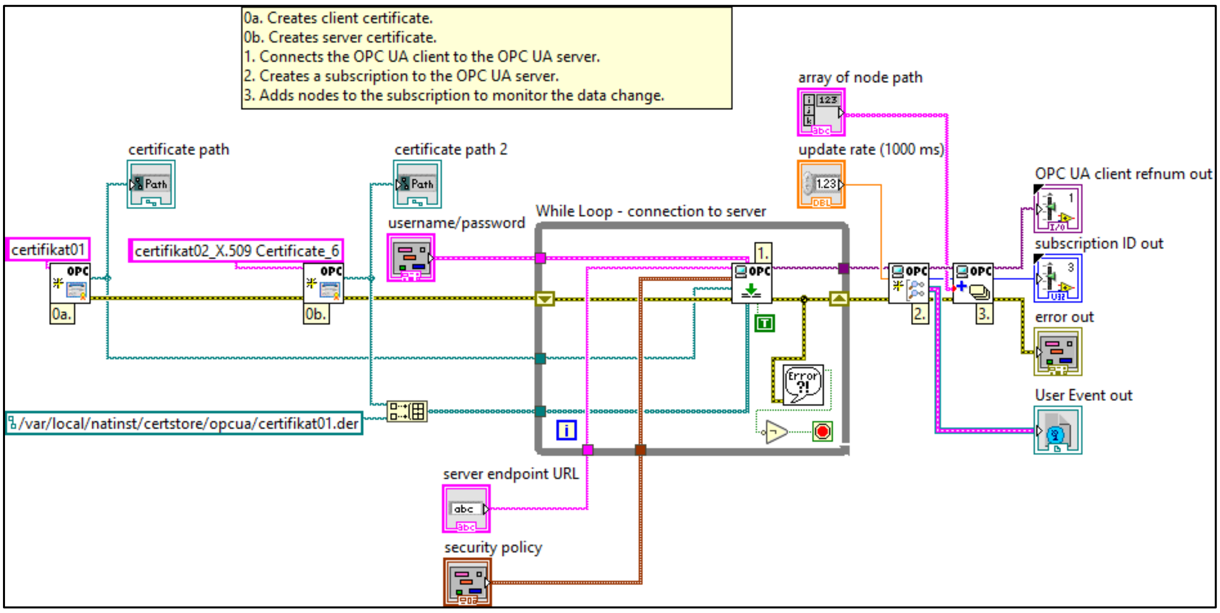


Figure 35 - Flat Sequence Structure for connection



0a. Creates client certificate.
0b. Creates server certificate.
1. Connects the OPC UA client to the OPC UA server.
2. Creates a subscription to the OPC UA server.
3. Adds nodes to the subscription to monitor the data change.

Figure 36 - Block diagram for "connecting to the OPC UA server" subVI

As can be noticed from the figure above, the **Connect.vi** is placed inside a **While Loop**. This was done because it is not always possible to establish communication from the first attempt. The stopping condition for the loop is the absence of error, that is achieved by using **Simple Error Handler.vi** and logical operation **not**.

After connection and node subscription has been established, the **communication loop** can be started (Figure 37). The loop contains a flat sequence structure with two frames to ensure that operations are executed sequentially in the right order.

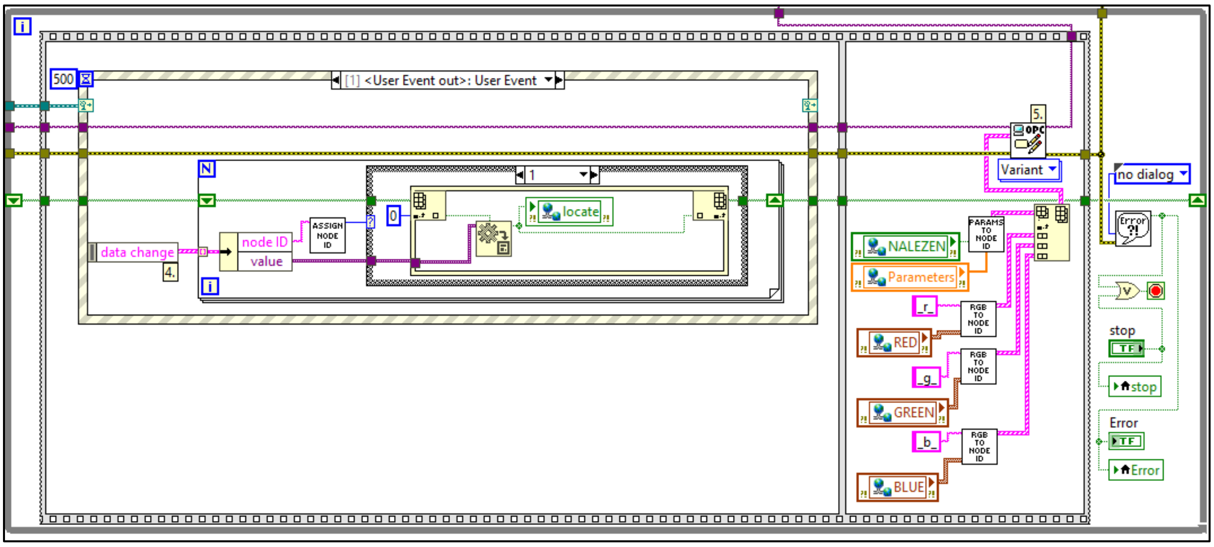


Figure 37 - Communication loop

In the first frame, the data change monitoring takes place using the **Register for Events** function wired to the “User event out” connector of the **connecting to the OPC UA server.vi** icon (label “OPC COM”). The data change is then wired to the **Event Structure**. There, conversion from data type **Variant** to actual format occurs. The principle of conversion is similar to the demo program. However, they differ in a number of monitored tags and the way how node ID (tag address) is associated with the case structure. In the demo program, an older version of the TIA Portal was used. There, node ID had format *Device.Block.Name_of_the_tag,* (for example *PLC_1.Memory.MyValue*) and could be directly associated with the required case (Figure 40). However, in the new version (V15) the tag format has changed to look like *ns=3;s="locate"*. Such a format cannot be associated with the case directly because of quotation marks.

To overcome this issue, another method was developed – node IDs of changed data are compared using **Equal?** functions. The Boolean outputs are then appended into a Boolean array using the **Build Array** function. After, the array is converted to an integer using **Boolean Array to Number** function. The value of the integer corresponds to a specific combination of the array. Thus, the output of the **Boolean Array to Number** function can be used as a case selector. All the described operations are placed in the subVI called **assigning node IDs** as demonstrated in Figure 39. SubVI’s icon is shown in Figure 38.

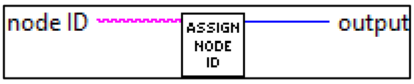


Figure 38 - Icon for “assigning node IDs” subVI

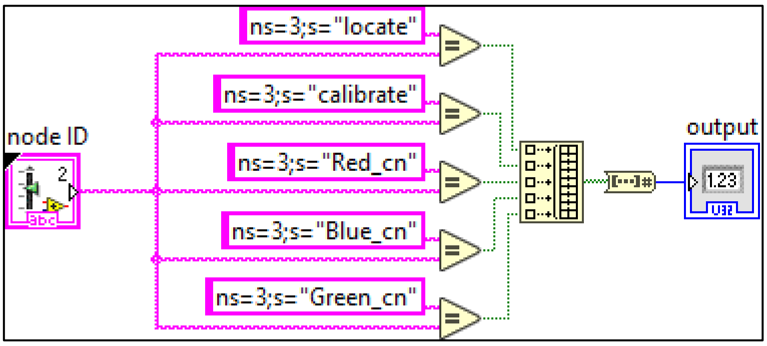


Figure 39 - Block diagram for “assigning node IDs” subVI

After the ID of the data change has been determined, the conversion to the appropriate format may occur. This operation is done using **Variant to Data** function in a **Case Structure**, where each case is designated for a specific tag. The way how the data is wired to the **Case Structure** was also modified – instead of wiring individual constants, an array was used. Wiring of the data to the array is accomplished using an **Array Index/Replace Elements** option of the **In Place Element Structure**. The whole process is placed in a **For Loop** as illustrated in Figure 41.

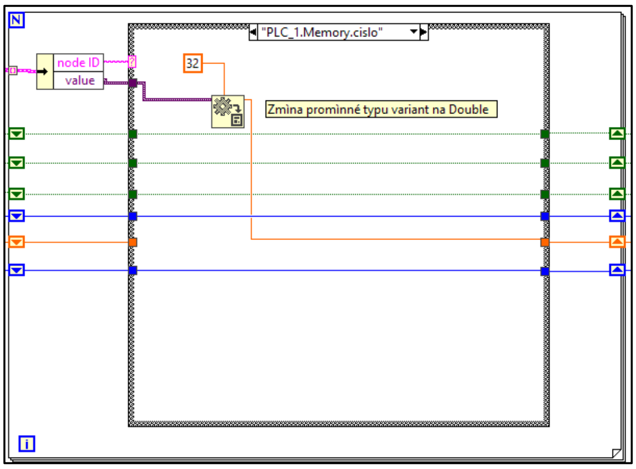


Figure 40 - Node ID monitoring before

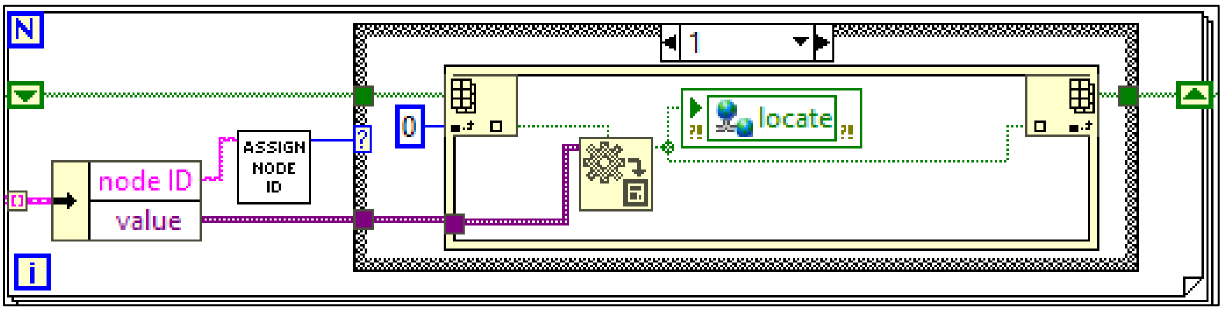


Figure 41 - Node ID monitoring after

In the second frame, writing of the data to the server takes place using **Multiple Write.vi** (Figure 42). This is another difference from the demo program – there, data was written separately to each tag using **Write.vi** (Figure 43). The reason for the difference is again the newer version of software used in this project.

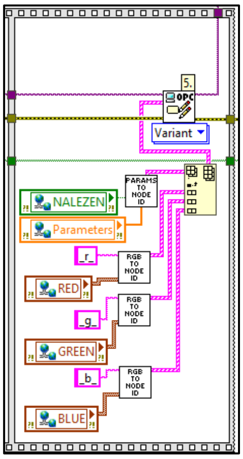


Figure 42 - Second frame

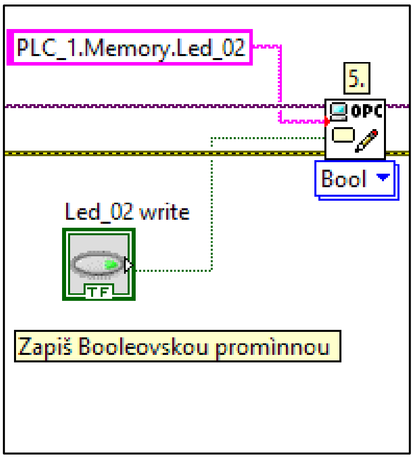


Figure 43 - "Write" VI

In order to write data using **Multiple Write.vi**, an array of **Requests** must be built and wired to the icon (Figure 44). The process for building a **Request** is following – data, converted to **Variant,** and node ID string is appended into a **Request** cluster using **Bundle by Name** function. An example is shown in Figure 45.

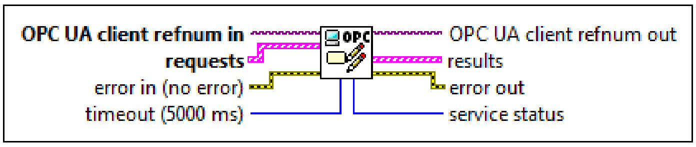


Figure 44 - Icon for "Multiple Write" VI

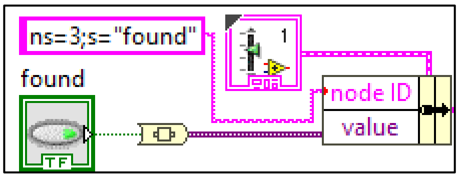


Figure 45 - Building request example

The process for building requests for data of the same type can be automated using the **Auto-Indexed Tunnel** of the **For Loop** – the data and node ID strings are appended to arrays and wired to the loop. As a result, the array of **Requests** is obtained. The loop is then situated in a **Flat Sequence Structure** to ensure the correct order of execution. Additionally, the whole process is placed in a subVI.

Two of such subVIs were created: one for writing the object parameters, called **Parameters to PLC nodes** (Figure 46), and the other for calibration, called **RGB to PLC nodes** (Figure 47). Their icons can be seen in Figure 48 and in Figure 49 respectively.

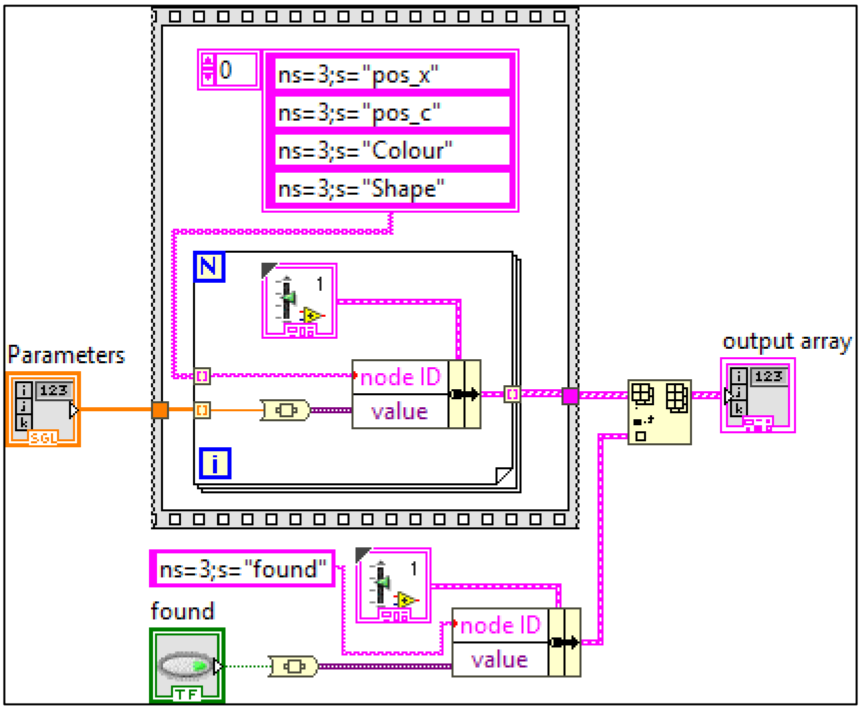


Figure 46 - Block diagram for “Parameters to PLC nodes" subVI

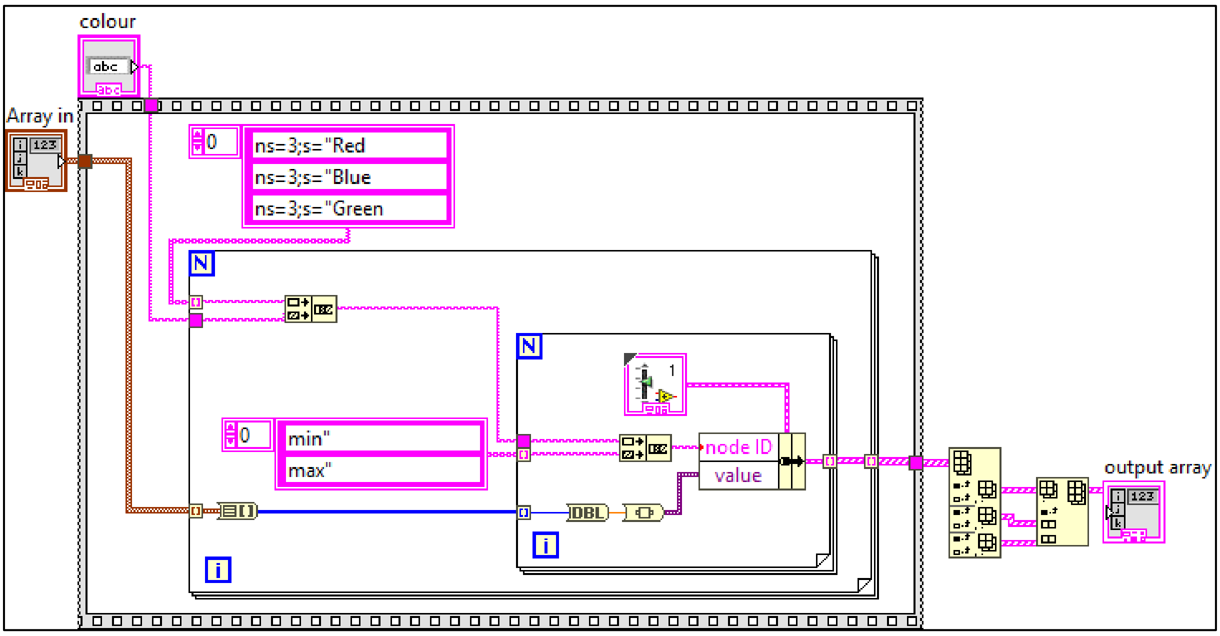


Figure 47 - Block diagram for "RGB to PLC nodes" subVI

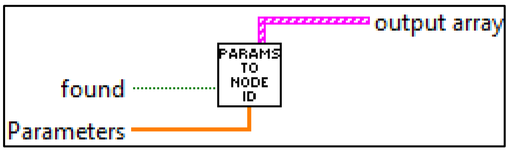


Figure 48 - Icon for "Parameters to PLC nodes" subVI

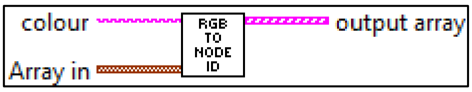
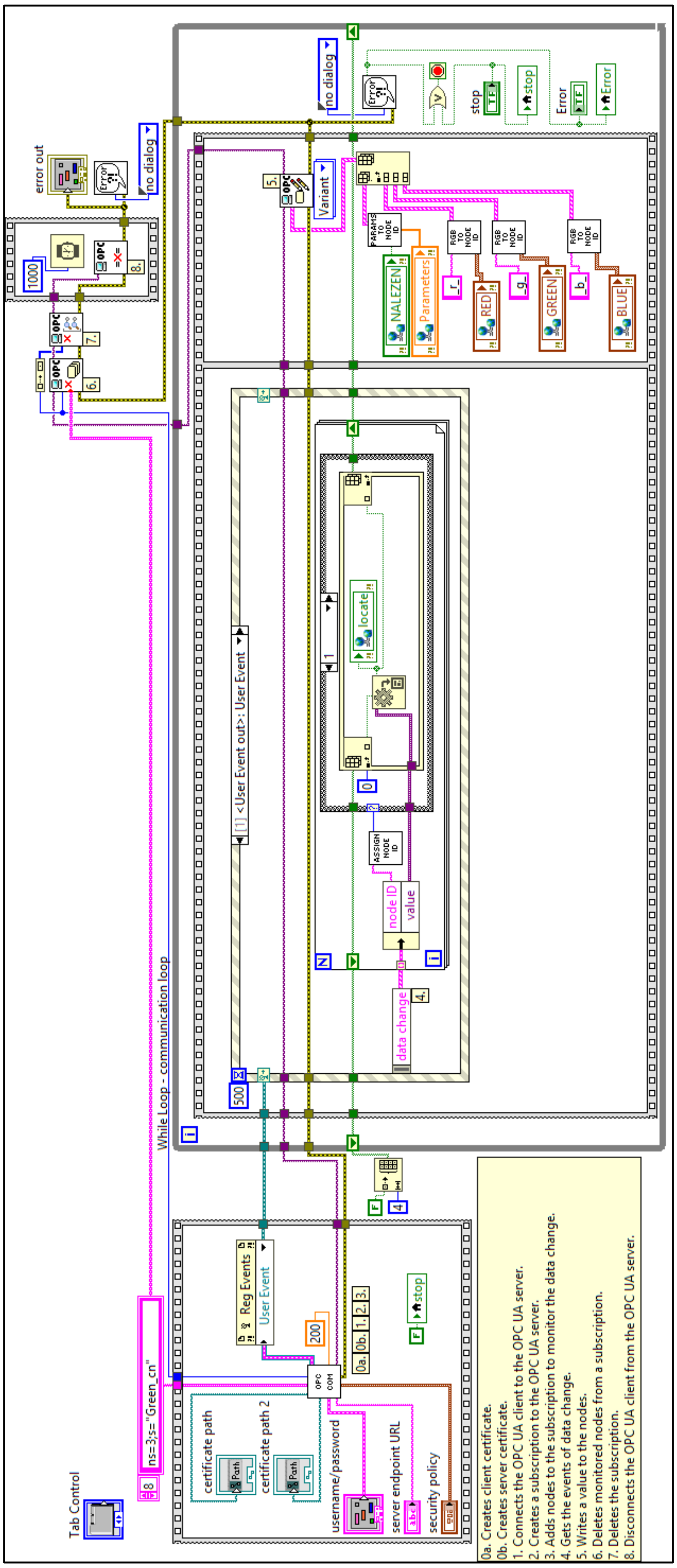


Figure 49 - Icon for "RGB to PLC nodes" subVI

The node monitoring and writing operation are in the **communication loop**. Thus, they are repeated for the whole duration of the program. The stopping condition of the loop is either press of the **STOP** button on the front panel or occurrence of an error. This information is also forwarded to the **inspection loop** in the form of **stop** and **Error** local variables.

After the **communication loop** has been stopped, it is necessary to disconnect the client from the OPC UA server. The process consists of 3 VIs: **Delete monitored nodes.vi** – deletes nodes from subscription; **Delete Subscriptions.vi** – deletes node subscriptions from the server; **Disconnect.vi** – disconnects client from the server. The execution of the last VI is delayed to ensure safe disconnection. The entire communication part of the program is shown in Figure 50.



0a. Creates client certificate.
0b. Creates server certificate.
1. Connects the OPC UA client to the OPC UA server.
2. Creates a subscription to the OPC UA server.
3. Adds nodes to the subscription to monitor the data change.
4. Gets the events of data change.
5. Writes a value to the nodes.
6. Deletes monitored nodes from a subscription.
7. Deletes the subscription.
8. Disconnects the OPC UA client from the OPC UA server.

Figure 50 - Communication part of the Inspection program

3.2.2.2 Inspection loop

The part of the program responsible for the inspection is shown in Figure 57 (see Attachment 2a for the full block diagram). The main difference from the original program is that object properties are appended to **Parameters** – the array type of **Shared** Variable. This was done to automate the building of **Requests** as explained in the previous section. This operation was placed in a subVI. In total, there are three subVIs in the **inspection loop**:

1. **building Parameters array**, labelled as "TO PARAMS ARRAY" – operations for the creation of **Parameters** array (see Figure 51 for block diagram and Figure 52 for the icon);
2. **unbundle and build array**, labelled as "UBN" – optimization (see Figure 54 for block diagram and Figure 54 for the icon);
3. **conversion**, labelled as "CONV" – also optimization (see Figure 55 for block diagram and Figure 56 for the icon).

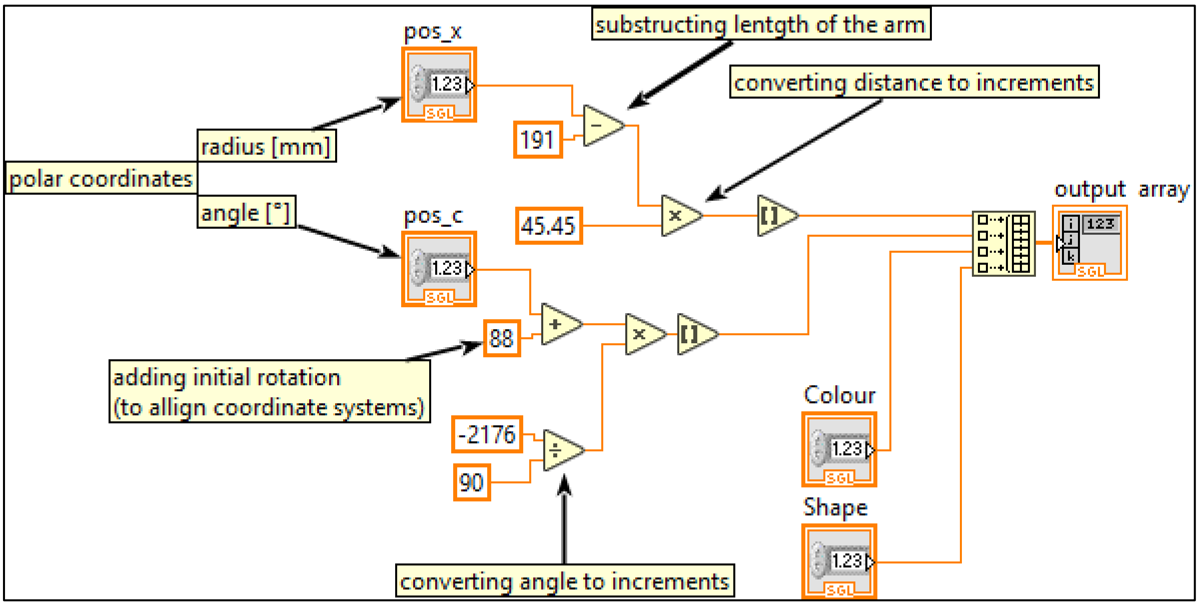

pos_x
substructing lentgth of the arm
converting distance to increments
1.23
SGL
191
radius [mm]
polar coordinates
angle [°]
pos_c
45.45
output array
123
88
adding initial rotation
(to allign coordinate systems)
-2176
90
Colour
Shape
converting angle to increments

Figure 51 - Block diagram for "building Parameters array" subVI

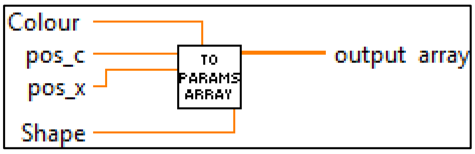


Figure 52 - Icon for "building Parameters array" subVI

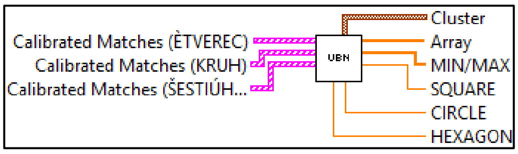

Calibrated Matches (ÈTVEREC)
Calibrated Matches (KRUH)
Calibrated Matches (ŠESTIÚH...
UBN
Cluster
Array
MIN/MAX
SQUARE
CIRCLE
HEXAGON

Figure 53 - Icon for "unbundle and build array" subVI

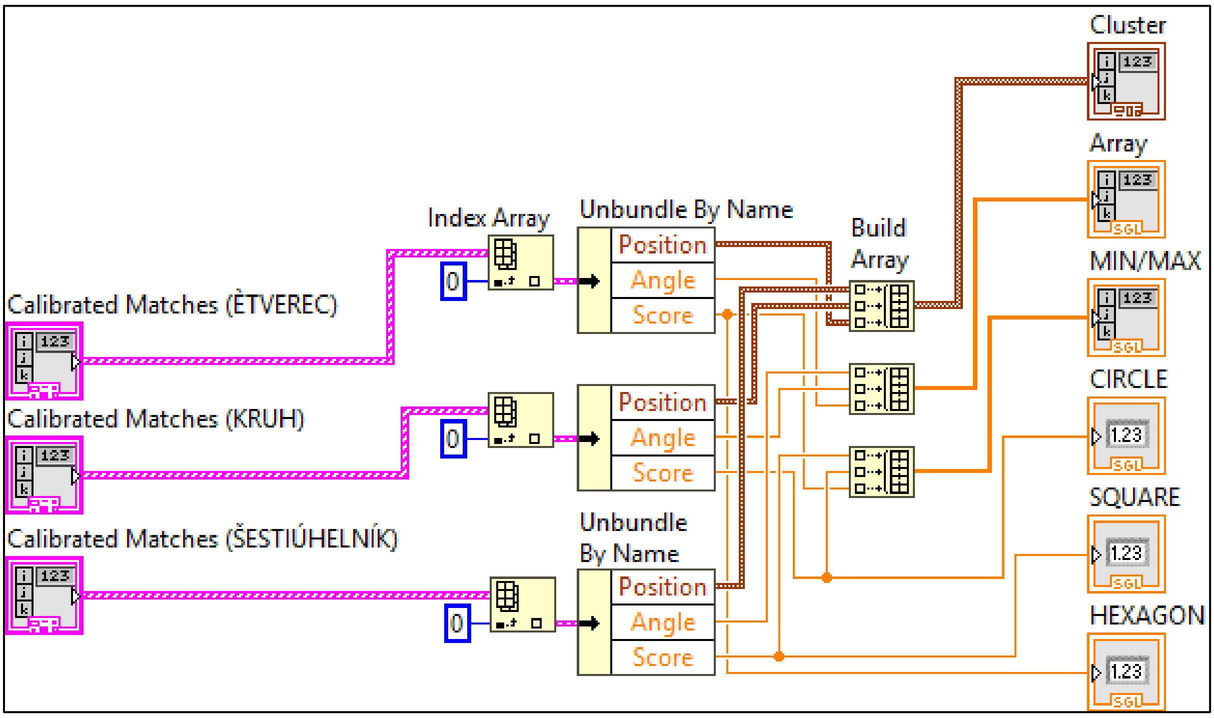


Figure 54 - Block diagram for "unbundle and build array" subVI

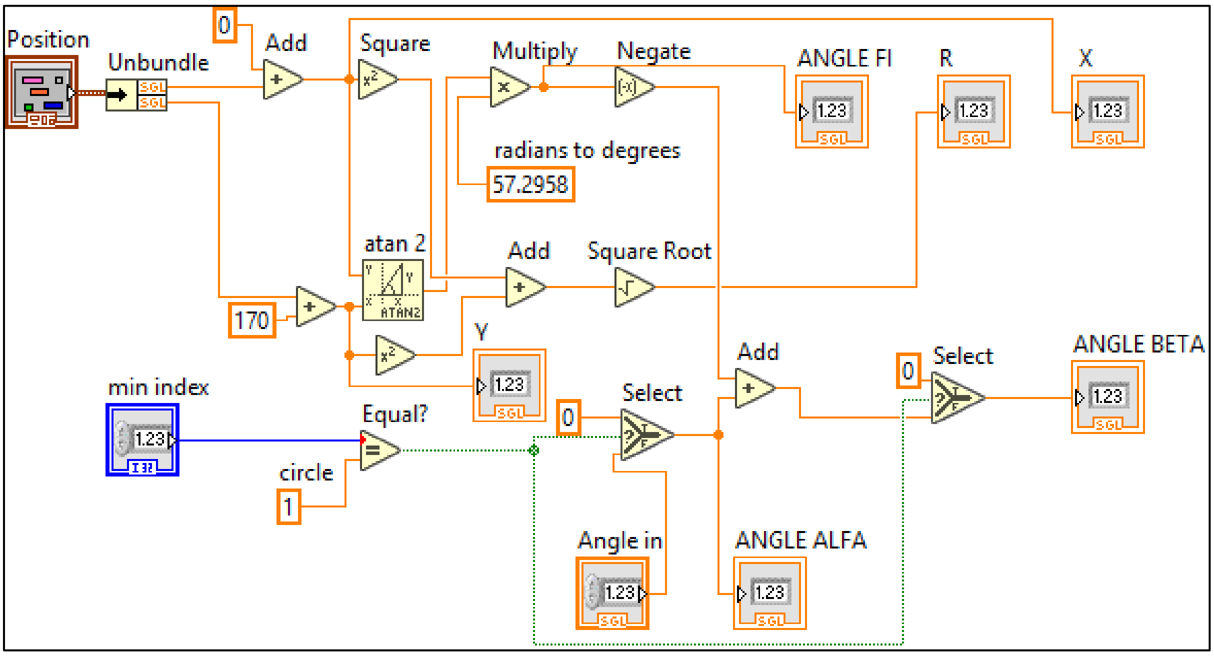


Figure 55 - Block diagram for "conversion" subVI

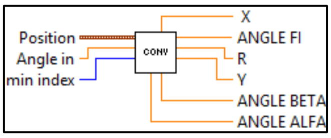
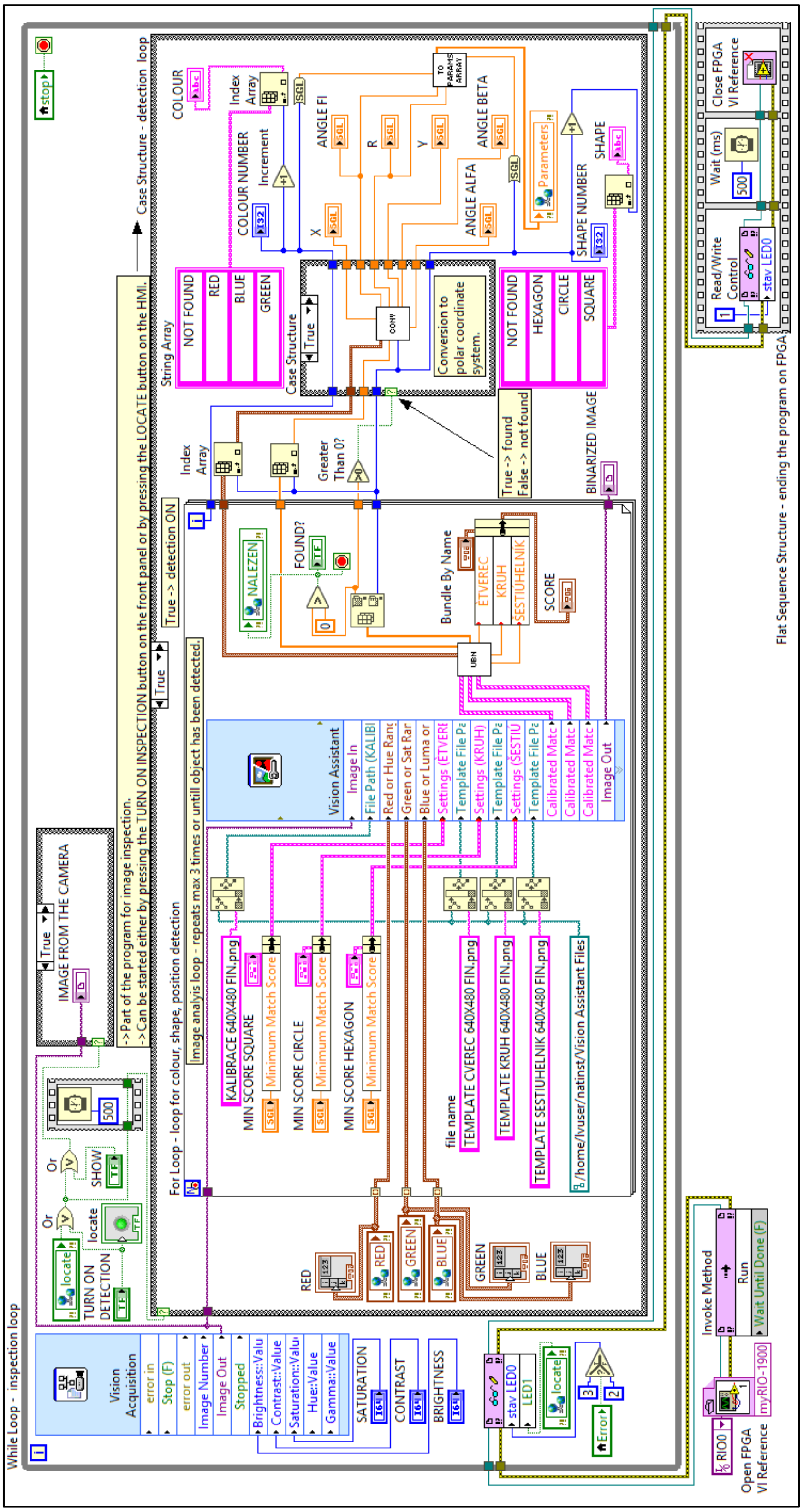


Figure 56 - Icon for "conversion" subVI



->Part of the program for image inspection.
->Can be started either by pressing the TURN ON INSPECTION button on the front panel or by pressing the LOCATE button on the HMI.

For Loop - loop for colour, shape, position detection

Image analysis loop - repeats max 3 times or untill object has been detected.

Conversion to polar coordinate system.

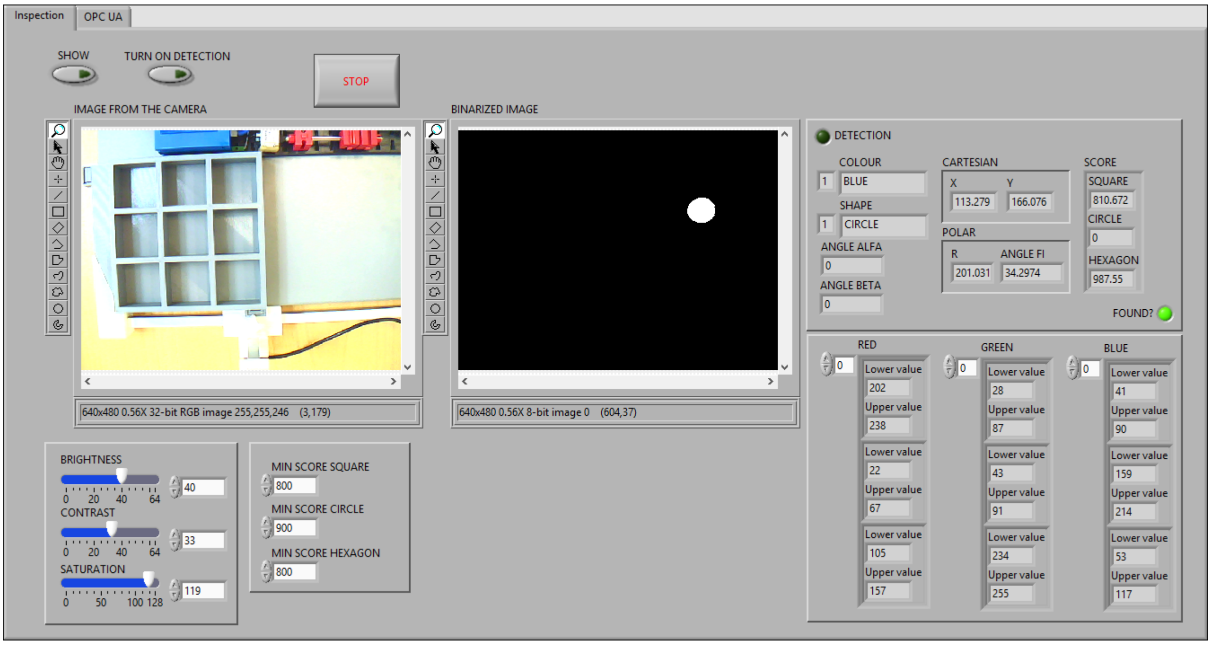
True -> found
False -> not found

Flat Sequence Structure - ending the program on FPGA

Figure 57 - Inspection loop

3.2.2.3 Front panel

For cleaner user interface front panel was divided into 2 tabs: Inspection and OPC UA. Inspection tab has the layout as Inspection program in Figure 15 with a single modification – labels were translated to the English language. OPC UA tab includes all the necessary controls for connection to the OPC UA server as demonstrated in Figure 59.



640x480 0.56X 32-bit RGB image 255,255,246 (3,179)

640x480 0.56X 8-bit image 0 (604,37)

COLOUR: 1 BLUE
SHAPE: 1 CIRCLE
ANGLE ALFA: 0
ANGLE BETA: 0

CARTESIAN: X 113.279, Y 166.076
POLAR: R 201.031, ANGLE FI 34.2974

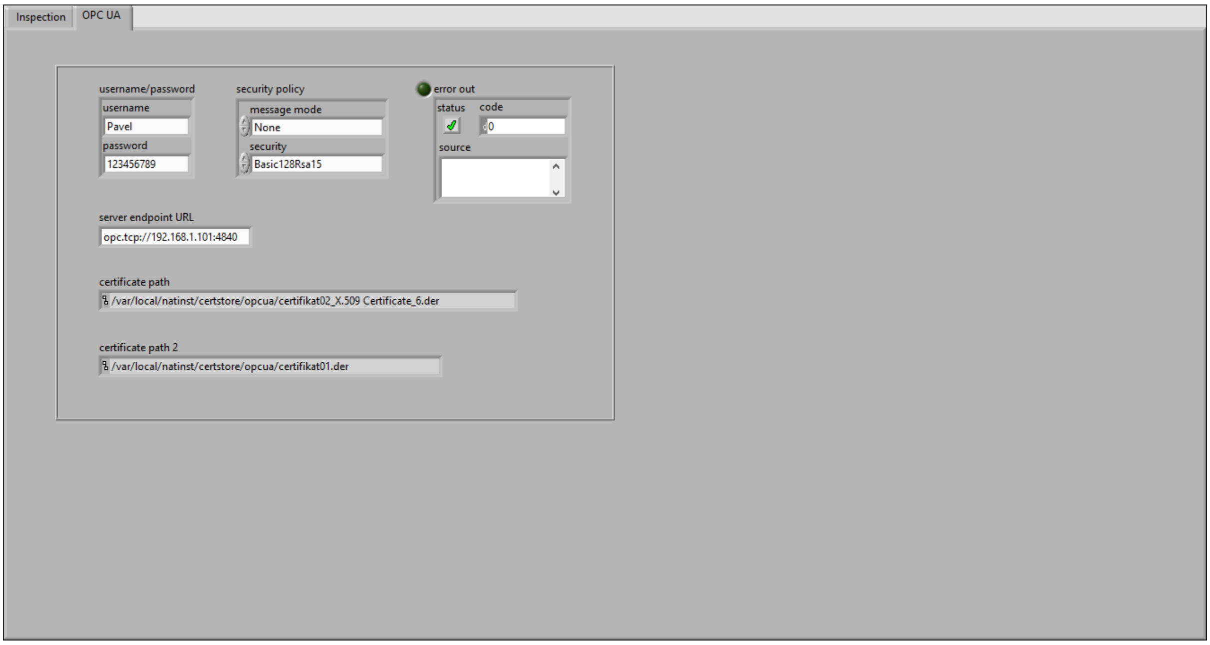
SCORE: SQUARE 810.672, CIRCLE 0, HEXAGON 987.55

RED		GREEN		BLUE	
0		0		0	
Lower value	202	Lower value	28	Lower value	41
Upper value	238	Upper value	87	Upper value	90
Lower value	22	Lower value	43	Lower value	159
Upper value	67	Upper value	91	Upper value	214
Lower value	105	Lower value	234	Lower value	53
Upper value	157	Upper value	255	Upper value	117

BRIGHTNESS 40
CONTRAST 33
SATURATION 119

MIN SCORE SQUARE 800
MIN SCORE CIRCLE 900
MIN SCORE HEXAGON 800

Figure 58 - Inspection tab



username/password
username: Pavel
password: 123456789

security policy
message mode: None
security: Basic128Rsa15

status ✓ code 0
source

server endpoint URL
opc.tcp://192.168.1.101:4840

certificate path
/var/local/natinst/certstore/opcua/certifikat02_X.509 Certificate_6.der

certificate path 2
/var/local/natinst/certstore/opcua/certifikat01.der

Figure 59 - OPC UA tab and Testing tab

3.3 Lighting

In order to improve the illumination of the workspace, a small LED searchlight was added to the camera stand. The light source is not specialized for the machine vision like described in the research part because it was not possible to find one suitable for fixing on the camera stand. Mounting realization can be seen in Figure 60.

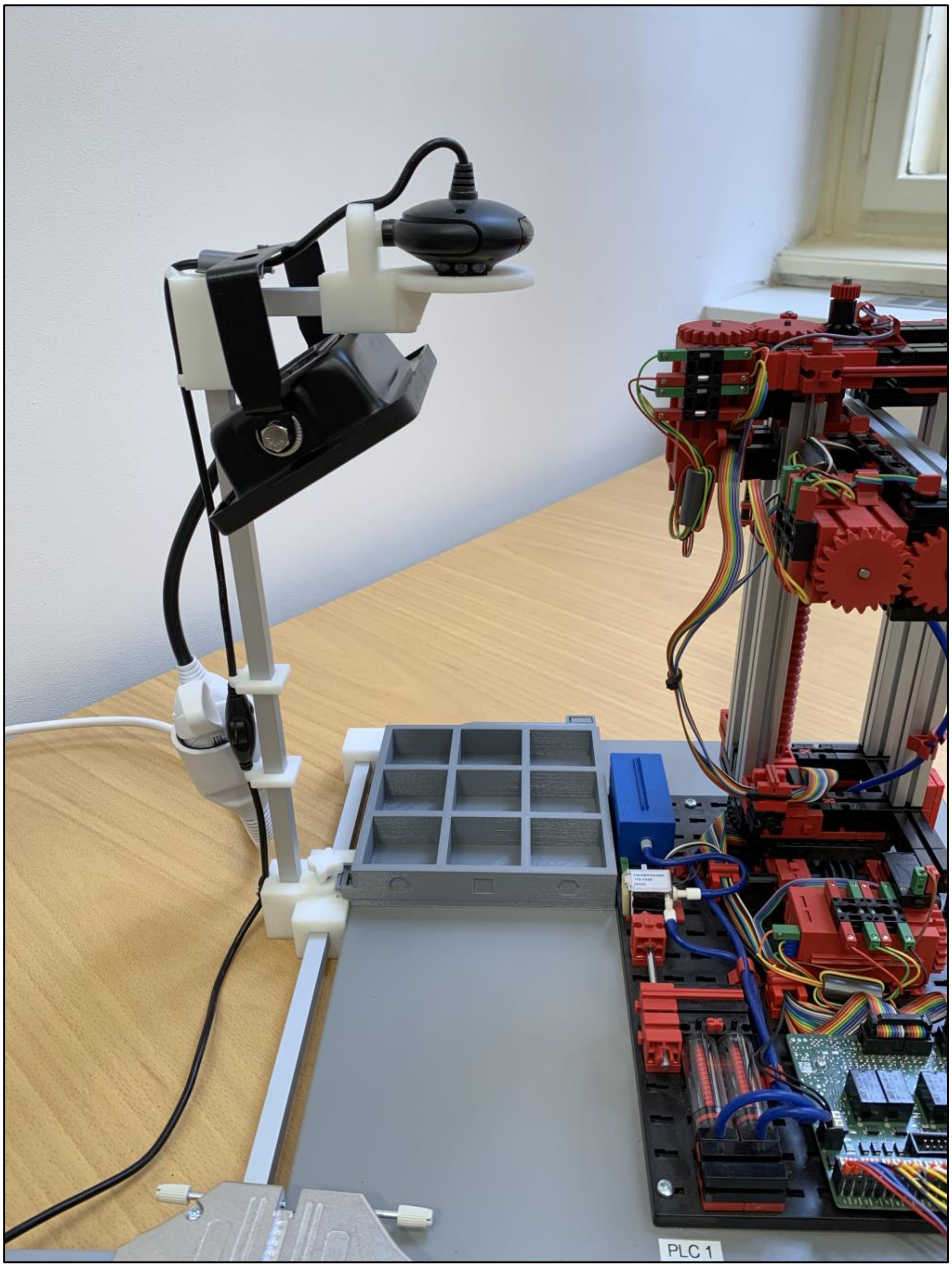


Figure 60 - Mounted light source

4 Testing the functionality

The functionality of the programs for PLC and myRIO was tested with the following results:

- OPC UA communication – 100% successful;
- Pick & place operation – 100 % successful;
- Lighting – unsuccessful.

As can be seen in the demonstration video (see attachment "Functional demo" on the CD), after the inspection program finds and analyses the object, PLC is able to navigate the manipulator to that object and then place to the required slot. The reason why hexagon shape was not used is because of the mistake in the storage design – slots are located too close so that manipulator cannot reach them. The proposed solution of illumination issue was not successful – the new light source illuminates the workspace too much due to the placement and intensity (Figure 61). As can be also seen from the demonstration video, the testing was performed under background illumination because it was found to be the most reliable. Thus, for proper functioning, it is necessary to perform colour calibration every time when the lighting conditions significantly change.

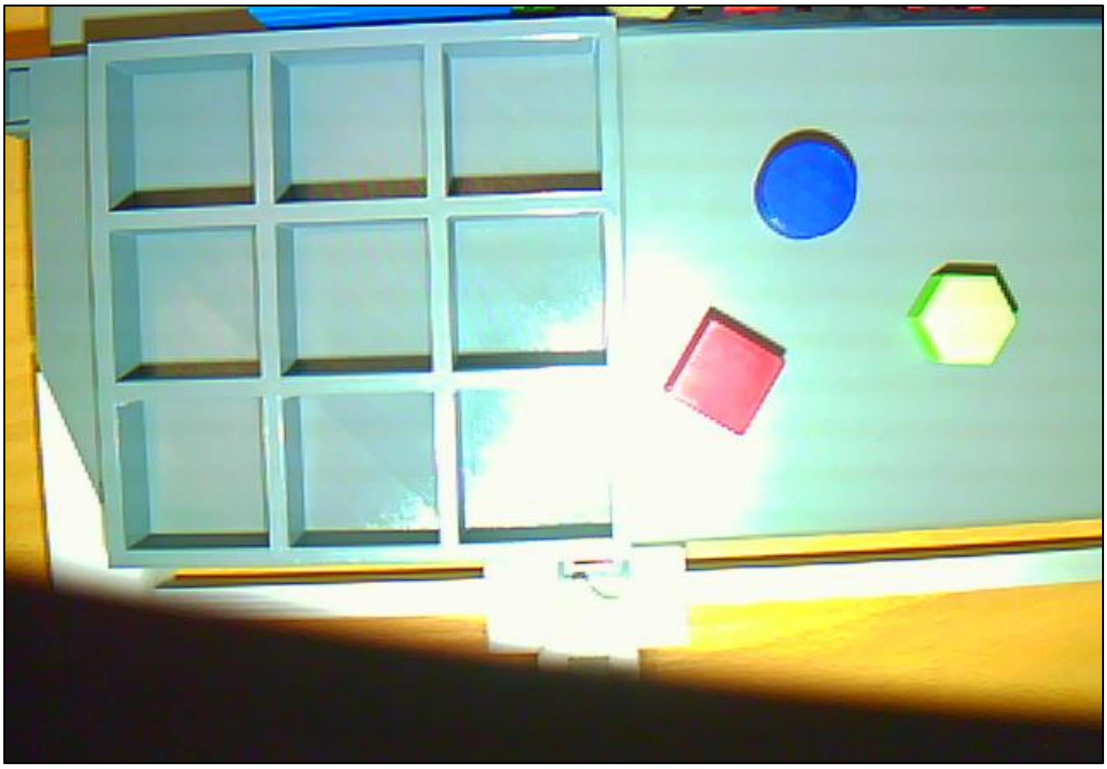


Figure 61 - Image from the camera when the additional light source is on

5 Alternative Solution – Calibration

After testing it was discovered that even with an additional light source illumination conditions cannot be kept constant. Thus, for the inspection program to work, RGB colour ranges have to be calibrated every time conditions change. Normally, the calibration requires a computer with LabVIEW and Vision Assistant software installed. Such a requirement is not convenient because the whole system should be able to run standalone. Hence, it was decided to implement additional functionality that would allow an easy calibration from the HMI screen.

It is necessary to set six parameters for each colour of the used shape. The parameters are upper and lower limits of RGB ranges (example of settings for green colour is shown in Figure 63). In total there are 18 parameters to be set. The original calibration method is described in [1].

The new calibration method also uses the Vision Assistant. There, a script with two steps is created:

1. *Image Mask* – to set the calibration region;
2. *Histogram* – to extract RGB parameters.

This script is placed in the block diagram in a form of Express VI: input to the VI is an image from the camera and the output is a 2D array. The array contains the data that can be used to construct a histogram like in Figure 63. The range between the lower and upper limits is approximately constant. Thus, the limits can be obtained by finding the index of the maximum value and adding/subtracting half of the range (chosen to be 60). However, not all the parameters can be set in this way – the shape’s colour own limits are constant. For example, for a green shape lower limit is 215 and the upper limit is the maximum possible (255). The calibration is implemented as a **calibration** subVI (see Figure 64 for full block diagram and Figure 62 for the icon).

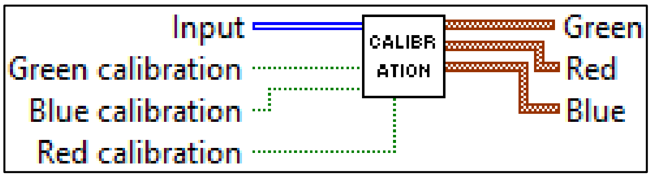
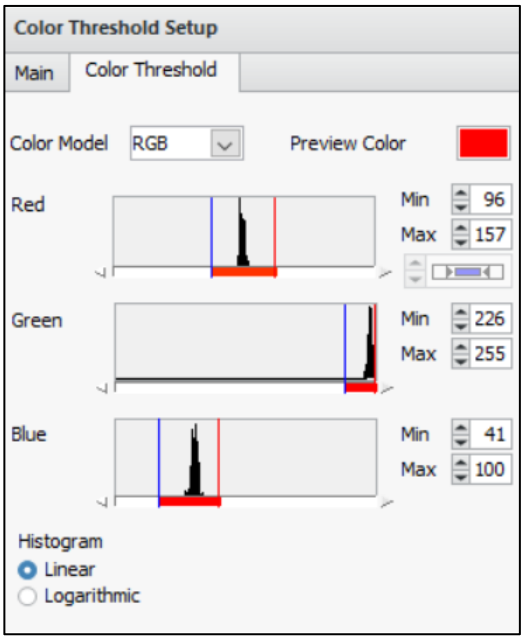


Figure 62 - Icon for "calibration" subVI



Color Threshold Setup

Main | Color Threshold

Color Model: RGB

Preview Color: 🟥

Red: Min 96, Max 157

Green: Min 226, Max 255

Blue: Min 41, Max 100

Histogram
● Linear
○ Logarithmic

Figure 63 - RGB settings example

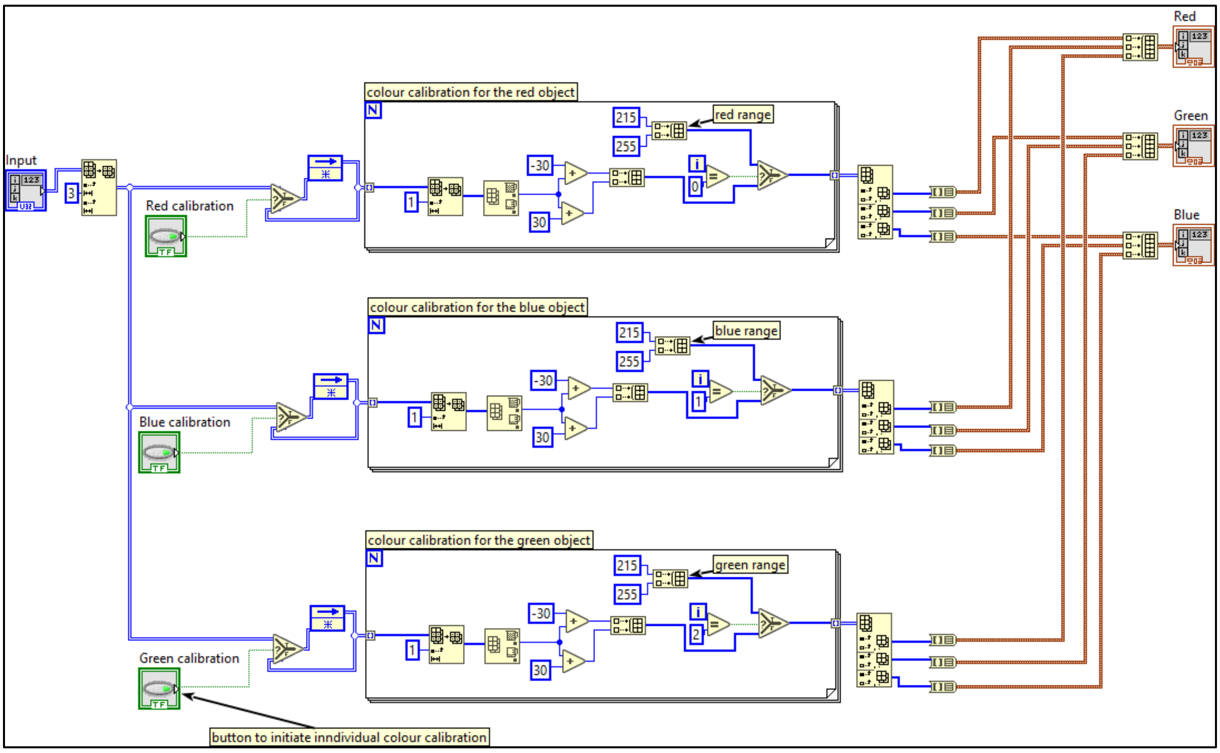

Input
3
Red calibration
colour calibration for the red object
N
215
255
red range
-30
30
1
i
0
Blue calibration
colour calibration for the blue object
215
255
blue range
-30
30
1
i
1
Green calibration
colour calibration for the green object
215
255
green range
-30
30
1
i
2
button to initiate inndividual colour calibration
Red
Green
Blue

Figure 64 - Block diagram for “calibration” subVI

In the **inspection loop**, calibration is implemented as a false case of the **detection loop** case structure (Figure 65). Additionally, an extra case structure is used to ensure that calibration is running only when required – **calibrate** shared variable is activated by the “Calibration” toggle on the HMI screen.

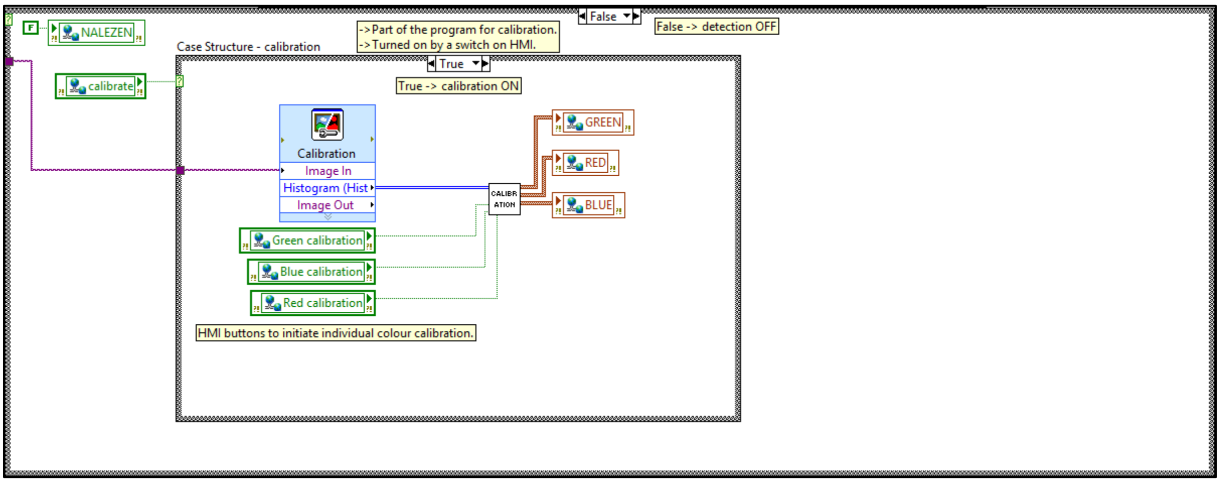
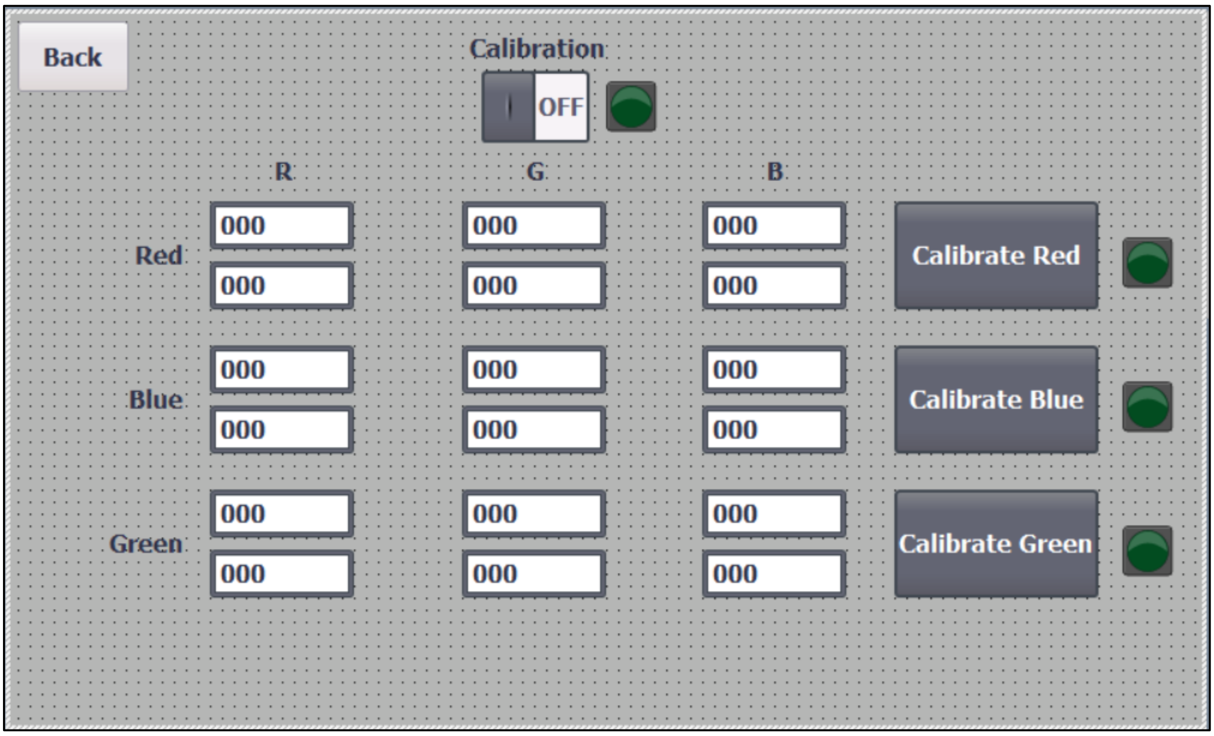


Figure 65 - Calibration implemented in the actual program

For user interaction, an extra HMI panel was created (Figure 66). The panel contains buttons to initiate the calibration as well as indicators to show the current settings. The way how the calibration is performed is the following:

1. Calibration mode is turned on;
2. Colour sample is placed on the designated spot;
3. Calibrate button for the appropriate colour is pressed until the values change;
4. Steps 2 and 3 are repeated for other colours;
5. Calibration mode is turned off.



	R	G	B
Red	000	000	000
	000	000	000
Blue	000	000	000
	000	000	000
Green	000	000	000
	000	000	000

Figure 66 - HMI screen for calibration

6 Conclusion

Communication possibilities between Siemens PLC and NI myRIO were mapped. The lighting options for the Fischertechnik were reviewed. The source codes for both the PLC and the vision system were edited so that the position, colour and shape of the object in the manipulator workspace are forwarded to the PLC. An example application was prepared, tested and the functional demo was recorded. Instructions for using the machine vision in PLC programming class were built (see Attachment 3b). Illumination issue was not resolved – instead, an alternative solution based on calibration was implemented.

Thus, the project assignment was fulfilled. In future, a fixture for the LED searchlight could be manufactured to resolve the illumination issue. Alternatively, an enclosure can be added to minimize the influence of the background illumination.

Bibliography

[1] JUST, Dominik. *Detekce polohy, orientace a vlastností dílců pro aplikace pick and place*. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze.

[2] *SIMATIC S7-1500 / ET 200MP Automation system In a nutshell*. Siemens AG, 2016. 109481357. Available at: https://support.industry.siemens.com/cs/document/109481357/simatic-s7-1500-et-200mp-automation-system-in-a-nutshell?dti=0&dl=en&lc=de-WW

[3] HENNING, Carl. THE DIFFERENCE BETWEEN PROFIBUS AND PROFINET. *Profinet* [online]. Scottsdale: PI North America, 2006-2019 [cit. 2019-02-23].

[4] BERGER, Hans. *Automating with SIMATIC S7-1500: Configuring, Programming and Testing with STEP 7 Professional*. Erlangen: Publicis Publishing, 2014. ISBN 978-3-89578-919-9.

[5] Seamlessly Connect to Third-Party Devices and Supervisory Systems. *National Instruments* [online]. b.r. [cit. 2019-05-18]. Available at: http://www.ni.com/cs-cz/innovations/white-papers/17/seamlessly-connect-to-third-party-devices-and-supervisory-system.html

[6] POWELL, James. Profibus and Modbus: a comparison. *Automation.com* [online]. b.r. [cit. 2019-02-21]. Available at: https://www.automation.com/automation-news/article/profibus-and-modbus-a-comparison

[7] OPC and OPC UA explained. *Novotek* [online]. Malmö: Novotek, 2019 [cit. 2019-02-23]. Available at: https://www.novotek.com/en/solutions/kepware-communication-platform/opc-and-opc-ua-explained

[8] Unified Architecture. *OPC Foundation* [online]. Scottsdale: OPC Foundation, 2019 [cit. 2019-02-23]. Available at: https://opcfoundation.org/about/opc-technologies/opc-ua/

[9] *OPC UA: The open interface for simple network integration*. Nuremberg: Siemens AG, 2018. Article-No. DFFA-B10293-02-7600. Available at:

https://assets.new.siemens.com/siemens/assets/public.1532340167.2e1b43b59229eba58e690 98f26a7c5e4f2461457.dffa-b10293-02-7600-opc-ua-flyer-72dpi.pdf

[10] HAVLE, Otto. Strojové vidění IV: Osvětlovače. *Automa* [online]. 2008, **2008**(4), 47-49 [cit. 2019-02-23]. Available at: http://automa.cz/Aton/FileRepository/pdf_articles/36988.pdf

[11] Plošné osvětlovače. *Analýza obrazu* [online]. Brno: ABBAS, a.s., 2011-2019 [cit. 2019-02-24]. Available at: http://www.analyza-obrazu.cz/osvetlovace/plosne/

[12] MARTIN, Daryl. A Practical Guide to Machine Vision Lighting. *National Instruments* [online]. Austin: National Instruments, 2019 [cit. 2019-02-23]. Available at: http://www.ni.com/white-paper/6901/en/

[13] Ring Lights. *Smart View* [online]. Otrokovice: Smart View s.r.o., b.r. [cit. 2019-02-24]. Available at: https://www.smartview.cz/en/illumination/ring-lights

[14] Tutorial: SubVIs. *National Instruments* [online]. 2018 [cit. 2019-05-24]. Available at: http://www.ni.com/tutorial/7593/en/

List of Figures

List of Tables

List of Attachments on the CD

PDF attachments:

1. Source Codes forTIA Portal
2. Source Codes for LabVIEW
3. Manuals

Programs:

1. Original programs
2. Modified programs

Video files:

1. Functional demo