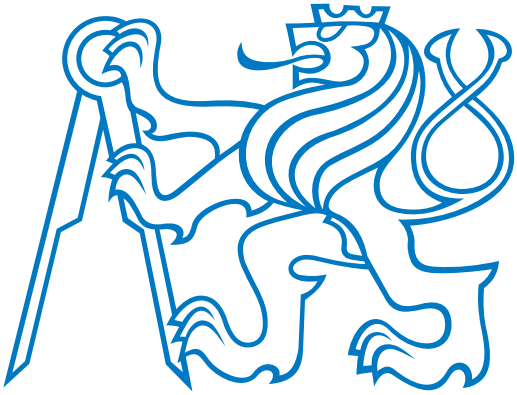
Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics
Study Programme: Cybernetics and Robotics
Field of Study: Robotics



Visual Functional Testing of Electronic Systems

Kamerová kontrola funkčnosti elektronických systémů

Master's Thesis

Author: Bc. Jan Paštyka
Supervisor: Ing. Vladimír Smutný, Ph.D.

May 2018

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Paštyka Jan** Personal ID number: **420297**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Robotics**

II. Master's thesis details

Master's thesis title in English:

Visual Functional Testing of Electronic Systems

Master's thesis title in Czech:

Kamerová kontrola funkčnosti elektronických systémů

Guidelines:

1. Get familiar with the basic methods of computer vision and image processing.
2. Design the system, which will capture images of front panel of industrial electronics.
3. Propose and implement the image processing tools which will check the status of LED indicators in the captured images.
4. Propose and implement the interface to the testing system and overall control of the system.
5. Test the system on the selected tested device and make conclussion.

Bibliography / sources:

[1] Milan Sonka, Vaclav Hlavac, and Roger Boyle: Image Processing, Analysis and Machine Vision. Thomson, 3rd edition, 2007.
[2] Bernd Jahne: Digital Image Processing. Springer Verlag, Berlin, 2005.
[3] Firemní literatura výrobců kamer: Basler, Allied Vision..
[4] Dokumentace sběrnice Modbus.

Name and workplace of master's thesis supervisor:

Ing. Vladimír Smutný, Ph.D., Robotic Perception, CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **09.01.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Vladimír Smutný, Ph.D.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master'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 master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Author statement for undergraduate thesis

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date .....

.....
signature

Acknowledgement

I would like to thank my thesis supervisor Ing. Vladimír Smutný, Ph.D. for his advice, support, and consultations when writing this thesis. I would also like to thank my family for all their support throughout my studies and writing this thesis.

Jan Paštyka

Annotation

This thesis deals with design and implementation of a system for automatic state recognition of diagnostic LEDs on industrial devices. LEDs states are recognized from images captured by an industrial camera. LED state analysis is performed on a computer connected to the camera and also to the test control system, which receives information about the LED state upon request. This thesis contains a description of a selection of appropriate industrial camera and lens, a design of a mechanical solution of a camera holder and description of an implemented user application. The application allows to configure a camera, performs LEDs states recognition, and communicates with the test control system. The thesis also describes the design of all parts of the system, results of its testing under real conditions, and a detailed user manual. The final system is sufficiently variable to be used during testing of standard types of industrial devices with green, red, and orange LED indicators. The main benefit of this thesis is the broadening of the possibilities of automation of software tests of industrial devices.

Keywords

computer vision, industrial camera, LED state, diagnostic LEDs, classification

Anotace

Tato práce se zabývá návrhem a realizací systému pro automatické rozpoznávání stavů diagnostických LED na průmyslových zařízeních. Stav LED diod je rozpoznáván ze snímků pořízených průmyslovou kamerou. Analýza stavu LED je prováděna na počítači, ke kterému je připojena tato kamera a nadřazený systém řídicí testy, kterému jsou na požádání posílány informace o rozpoznaných stavech LED. Součástí této práce je popis výběru vhodné průmyslové kamery a objektivu, návrh mechanického řešení uchycení kamery na stojan s testovanými zařízeními a implementace obslužné aplikace. Tato aplikace umožňuje konfiguraci kamery, provádí rozpoznávání stavů LED a zajišťuje komunikaci s nadřazeným systémem, který řídí test. V práci je popsán postup návrhu všech dílčích částí systému, výsledky jeho testování za reálných podmínek a také podrobný uživatelský manuál. Výsledný systém je dostatečně variabilní pro použití při testování standardních typů průmyslových zařízení se zelenými, červenými a oranžovými indikačními LED. Hlavním přínosem této práce je rozšíření možností automatizace softwarových testů průmyslových zařízení.

Klíčová slova

počítačové vidění, průmyslová kamera, stav LED, diagnostické LED, klasifikace

Contents

List of Figures

List of Tables

Glossary

A-KAZE Accelerated-KAZE. Algorithm for feature detection and description.

API Application Programming Interface. Set of tools used for communication between different software components.

ATE Automatic Test Equipment. A system that performs tests of device under test (DUT). DUT is controlled based on these tests, and its response is measured and analyzed.

BRIEF Binary Robust Independent Elementary Features. Algorithm for feature description.

BRISK Binary Robust Invariant Scalable Keypoints. Algorithm for feature detection and description.

CCD Charge-Coupled Device. A device that converts the energy of incident light into electrical energy.

CPU Central Processing Unit

DUT Device Under Test. A developed device which is tested by automatic test equipment. In this thesis, it is typically a factory device with diagnostic LEDs.

FAST Features from Accelerated Segment Test. Algorithm for Corner Detection.

GUI Graphical User Interface. An interface between an application and a user which uses graphical icons and indicators for application control.

ID Identifier. Label that uniquely identifies an object.

IP Internet Protocol. A protocol which is used for transmitting data between computers by using the internet.

IR Infrared Radiation. Electromagnetic radiation with a longer wavelength than a wavelength of visible light.

JSON JavaScript Object Notation. A format for storing and exchange of data.

k-NN k-Nearest Neighbors. Classification method.

KAZE Japanese word for *wind*. Algorithm for feature detection and description.

LED Light-Emitting Diode

LUT Lookup Table. A data structure that stores data that would otherwise have to be computed. These data are accessed using a key.

ORB Oriented FAST and Rotated BRIEF. Algorithm for feature detection and description.

POSIX Portable Operating System Interface. Unified application interface that ensure portability of applications across different UNIX and other operating systems.

RAM Random-Access Memory

PWM Pulse-Width Modulation. Method of controlling the output device.

RANSAC Random Sample Consensus. Method of estimating model parameters from measured data.

SIFT Scale-Invariant Feature Transform. Algorithm for feature detection and description.

SMD Surface-Mount Device. An electronic component designed for surface mounting of printed circuit board.

SURF Speeded-Up Robust Features. Algorithm for feature detection and description.

USB Universal Serial Bus. The bus used to connect peripherals to a computer.

XML Extensible Markup Language. A format for storing and exchange of data.

Chapter 1

Introduction

Testing is an integral part of each development process. Many segments of a software testing are currently automated and technologies are moving towards complete automation of testing. There already exist many solutions for the simulation of user interaction. For example electronically controlled devices for pressing buttons on a device under test (DUT) or devices for simulation of a user-controlled touchscreen. Simulation of a user interaction is typically much more straightforward to implement than the interpretation of device state based on its observation.

One part of the testing of the industrial devices involves checking if a states of diagnostic LEDs on a DUT corresponds to its expected internal state. Mostly this test has to be done visually by a tester , but there are also automatic solutions. There can be used a device with a matrix of photoelements, which can be deployed on a DUT so that the photoelements are directly in front of the diagnostic LEDs. This type of devices can provide information about the state of diagnostic LEDs. The significant disadvantage of this approach is that this device cannot be used for DUTs with a different layout of LEDs. Another problem is that LEDs are typically entirely occluded by the device, and therefore they cannot be seen by a tester. Both these disadvantages can be eliminated by using a camera for recognition of LED state instead of the matrix of photoelements. Even though this approach is less reliable, it seems to be a better option in most of the typical cases.

The goal of this thesis is to design and implement a system for automatic recognition of the state of diagnostic LEDs of DUT and pass this information to the test control system. State of LEDs will be evaluated based on a DUT image captured by an industrial camera. The system should be easily configured by a user through an application with GUI.

Thesis Outline

- **Chapter 2** introduces requirements for the developed system and parameters of devices under test.
- **Chapter 3** is dedicated to designing and implementation of the whole system.

This chapter provides a detailed description of a mechanical solution and the selections of an industrial camera and appropriate software tools. Rest of the chapter is dealing with used techniques of image processing and implementation of the control application.

- **Chapter 4** contains detailed user's manual for the developed system.
- **Chapter 5** focuses on testing of the whole system under real conditions with a discussion of achieved results.
- **Chapter 6** contains a conclusion including restrictions and possible improvements of the developed system.

Chapter 2

System Requirements and Specifications

The whole system for LED state recognition is created according to the requirements defined by software testers from the company Siemens who are potential users of the system. Beside requirements to be met by the system, there are also specifications of the DUTs. Knowledge of all parameters and restrictions is necessary for finding a suitable solution.

Parameters of Devices Under Test:

- The width of observed area does not exceed 500 mm.
- The height of observed area does not exceed 300 mm.
- Front panels of devices have a shape of a rectangle.
- Observed LEDs can be round or rectangular.
- Width (or diameter) of observed LEDs is at least 3 mm.
- Each LED can be green, red, or orange.

System Requirements:

- The system shall be controlled by an application with graphical user interface (GUI).
- The system shall provide the current recognized state of the observed LEDs upon request of the test control system.
- The time duration between sending a request for current LED states by the test control system and receiving a response shall be less than 0.5 s.

- Communication between computer performing the analysis and the test control system shall use the Modbus protocol.
- An interface for communication between an application and the test control system on client side shall use the library *libmodbus 1.2.0*.
- Computer performing the analysis shall be connected to the test control system via Ethernet.
- In case of closing communication channel by a client application, a server shall wait for reconnection to a client.
- The distance between DUT and an industrial camera shall be maximally 600 mm.
- The system shall be able to recognize at least four states of LED: green, red, orange, and turned off.
- The system shall be able to suppress an influence of different lights conditions.
- The system shall be easy to use for LEDs recognition on different types of devices.
- Wiring of the system shall be straightforward.
- The system configuration shall be simple.

Chapter 3

System Design and Implementation

3.1 System Design

System design is based on requirements in Chapter 2. It could be split into three, almost independent, parts: mechanical solution, selection of appropriate hardware, and design of a software. First two parts are described in this section and software design is described in Section 3.4.

3.1.1 Function Principle

Block diagram of the whole system is shown in Fig. 3.1. Individual components of the system are described in more detail in the following paragraphs.

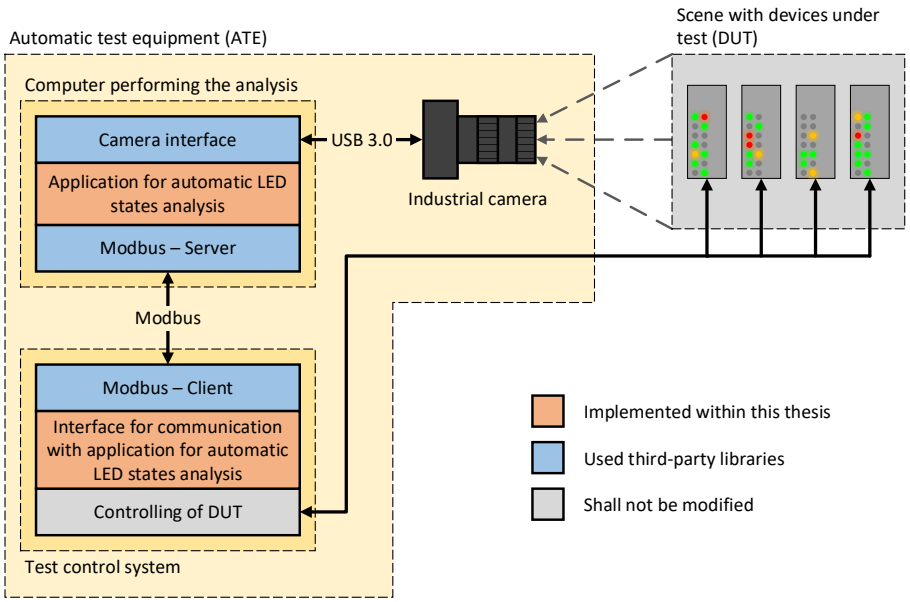


Figure 3.1: A simplified block diagram of the system. Red blocks are fully implemented within this thesis (by using third-party libraries). Blue blocks illustrate libraries essential for the system functionality. Gray block is a current test system, which shall not be modified.

Test setup consists of two separate parts: automatic test equipment (ATE) and devices under test (DUT). ATE is any set of devices performing tests on other devices. ATE currently consists only of a block "Controlling of DUT", but as it is shown in the diagram, it will be extended with the system for automatic recognition of LED states. DUT is a tested device which is controlled and checked by the test control system.

Camera Interface

An interface between industrial camera and application for LED state analysis uses camera API. Some information about the API is written in Section 3.2.2. Use of the API is in the more details described in Section 3.4.2.

Application for Automatic LED State Analysis

User application for automatic LED state analysis is called *LED State Analyzer*. This application is the main part of the system. It provides functionality for controlling a camera, processing of captured images, recognition of LED states, and interaction with the test control system. In the *LED State Analyzer* application, it is also possible to configure the type of DUT, LED state recognition, and communication with the test control system. Design of this application is described in Section 3.4.

Modbus Interface (Client, Server)

Modbus communication protocol is used for communication between *LED State Analyzer* and the test control system. This protocol is client–server based. *LED State Analyzer* is the server and the test control system is the client. Basic information about Modbus protocol is written in Section 3.2.3. Data transmission between the client and the server and its data structure is described in Section 3.4.6.

Interface for Communication with Application for Automatic LED State Analysis

Interface for communication with *LED State Analyzer* provides functionality for getting the current recognized states of LEDs. This interface differs for distinct configurations of DUTs. Interface description and structure of data transmitted between client and server is described in Section 3.4.6.

Controlling of DUT

Controlling of DUT is responsible for performing tests according to the configuration. DUTs are controlled using test control logic, which depends on the type of test and type of DUT. Controlling of DUT can use an interface for communication with *LED State Analyzer* but it is not allowed to implement any functions related to LED state recognition inside this part of test control system.

3.1.2 Mechanical Solution

Devices under test are placed on a rack. A mechanical construction with a camera is attached to the top of the rack. This construction is made of aluminum profiles with an adjustable arm on which an industrial camera is located. The revolute joint with lock allows the arm to be turned aside if necessary. This feature is useful when no tests are running, and the tester shall works on the rack without any restrictions. The concept of a mechanical solution is shown in Fig. 3.2.

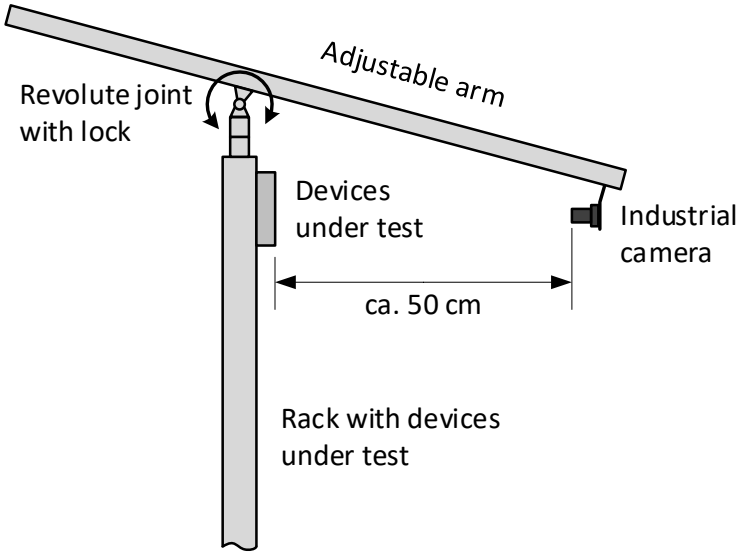


Figure 3.2: The concept of a mechanical solution of camera holder.

Photos of some parts of a mechanical construction are shown in Fig. 3.3, 3.4 and 3.5.

Figure 3.3: Attaching of a construction to the rack.

Figure 3.4: Mounting of the camera to the adjustable arm.

Figure 3.5: Revolute joint with lock.

3.1.3 Selection of the Industrial Camera

The *Basler daA2500-14uc* camera was chosen for implementation of the system (Fig. 3.6). This camera is designed and optimized for machine vision. Some of its parameters are written in Tab. 3.1. Decisions which led to the selection of this camera are described below.

Figure 3.6: Industrial camera *Basler dart daA2500-14uc* [1].

Table 3.1: Parameters of an industrial camera *Basler dart daA2500-14uc* [2].

Parameter	Value
Sensor Type	ON Semiconductor MT9P031
Sensor Size	5.7 × 4.3 mm
Resolution (H × V)	2592 × 1944 pixels
Resolution	5 MP
Frame Rate	14 fps
Mono/Color	Color
Interface	USB 3.0

It was necessary to consider all system requirements during selection of a suitable camera. It is required to recognize LEDs of different colors, so the camera has to be a color one. A frame rate of the camera has to be at least 2 fps. The camera should contain software trigger which enables to control the time of image capture. In a computer vision, it is not appropriate to use a camera with autofocus, and it is always better to use a separate lens with manual focus. Images from the camera should be of a very high quality without any lossy compression. The camera should also communicate via commonly used interface (e.g., USB or Ethernet) and provide sufficient support for Linux and C++ API (see Section 3.1.6). All these requirements are met by the selected camera.

Last important factor in the selection of a camera was its spatial resolution. Camera *Basler dart daA2500-14uc* uses Bayer filter, like most of the single-chip cameras. As it is shown in Fig. 3.7, in Bayer filter, half of all pixels are used for capturing green color and a quarter of all pixels is used for capturing red and blue color. Since recognition of colors is essential in this application, the minimum acceptable size of LED in the image must be decided. The following consideration was used for determination of minimum width (height) of the LED in pixels. Based on Shannon's Theorem, at least two times more pixels (linearly) should be used than necessary for color detection, which is two pixels. In practice, this minimum is not sufficient, and at least three pixels should be used. As it was already mentioned, a camera uses a Bayer mask, and therefore every set of four pixels contains only one pixel with information about red and blue color. It means that we need at least four times more pixels (twelve). There should also be some reserve for a possibility of inaccurate localization of LEDs or mild camera vibrations, so three more pixels are added. Based on these considerations, each LED should be displayed on an area with size at least 15 × 15 pixels.

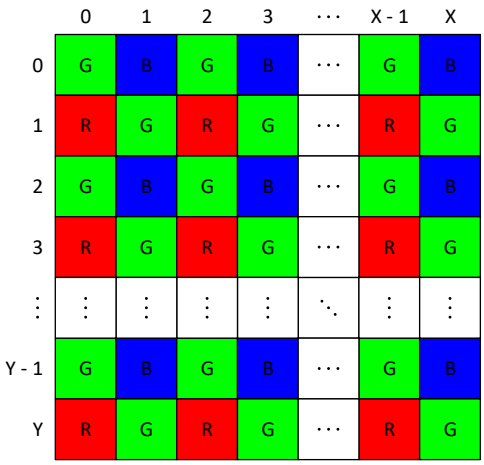


Figure 3.7: Bayer filter [3].

According to the DUT parameters (see Chapter 2), the minimum dimensions of each LED is 3 mm × 3 mm. Each LED should thus be visible on an area with the size of at least 15 × 15 pixels. Maximal dimension of a scene is 500 mm × 300 mm, and an aspect ratio of the image is 4:3, so it is sufficient to determine the minimal width (number of pixels in one row).

Minimal number of pixels in one row for the worst case (the broadest scene with the smallest LEDs) can be calculated using Eq. 3.1, where s_p is a minimal number of pixels in one row of a captured image, l_p is a width of the smallest LED in pixels, s_m is a maximal width of scene in millimeters, and l_m is a width of smallest LED in millimeters.

$$s_p = \frac{l_p \cdot s_m}{l_m} = \frac{15 \cdot 500}{3} = 2500 \text{ pixels} \tag{3.1}$$

The width of an image captured by a camera should be at least 2500 pixels based on the result of Eq. 3.1. Industrial camera *Basler dart daA2500-14uc* has resolution 2592 × 1944 pixels which is sufficient.

Spectral Response of the Camera

Camera *Basler dart daA2500-14uc* uses sensor MT9P031 manufactured by the company ON Semiconductor. An important property of a camera sensor is its spectral response, which is shown in Fig. 3.8. The spectral response curve excludes lens characteristics, light source characteristics, and IR cut filter characteristics [2]. This curves will be taken into account in image processing and LED color recognition.

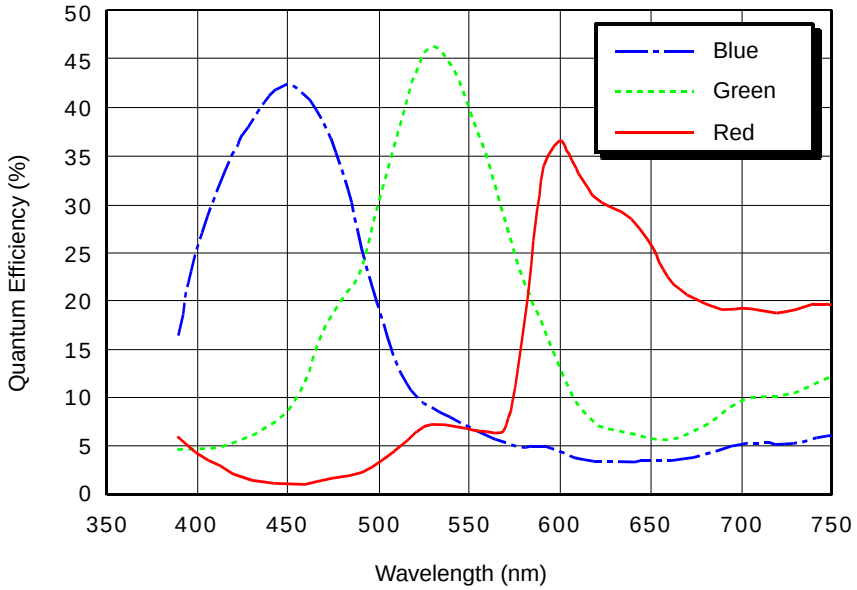


Figure 3.8: Spectral response of camera *Basler dart daA2500-14uc*[2].

3.1.4 Design of an Optical Path

The design of an optical path is based on system requirements in Chapter 2. It is necessary to know the size of an observed object, the distance between the camera and an observed object and size of the camera sensor to select a suitable lens. The camera with lens and an area it observes are illustrated in Fig. 3.9.

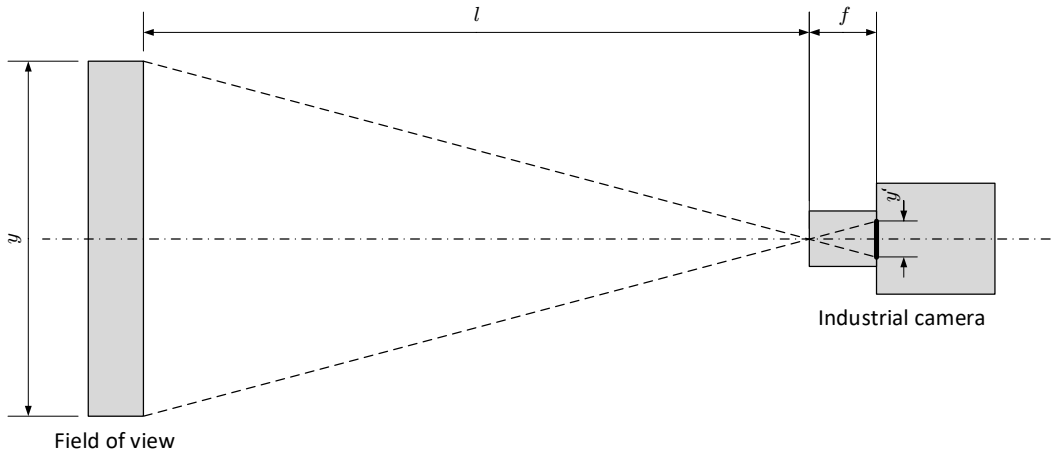


Figure 3.9: Illustration of a camera and an area it observes.

Based on this illustration we can compose the Eq. 3.2 for focal length calculation. Definitions of symbols: f is the focal length, l is the distance between observed object and lens, y is the width of an object (field of view) and y' is the width of a sensor. All dimensions are in millimeters.

$$f = \frac{l \cdot y'}{y} = \frac{600 \cdot 5.7}{500} = 6.84 \text{ mm} \tag{3.2}$$

Considering the result of Eq. 3.2 and available lenses for the camera *Basler dart daA2500-14uc* we can select lens with focal length 6 mm or 8 mm. Calculated focal length is valid for a maximum allowed distance between object and lens. This distance can be smaller in practise, and therefore it seems to be more appropriate to select a lens with focal length 6 mm. We selected *Basler Lens C125-0618-5M F1.8 f6mm* and its photo is shown in Fig. 3.10 [4].

Figure 3.10: Lens *Basler Lens C125-0618-5M F1.8 f6mm*.

3.1.5 Wiring Diagram

Wiring of the whole system is straightforward, which is also one of the requirements in Chapter 2. *LED State Analyzer* (which controls the camera, communicates with the test control system and performs image processing) is running on a separate computer. This computer is connected to the test control system via an ethernet cable and also to the industrial camera via a USB 3.0 cable. Wiring diagram of these connections is shown in Fig. 3.11.

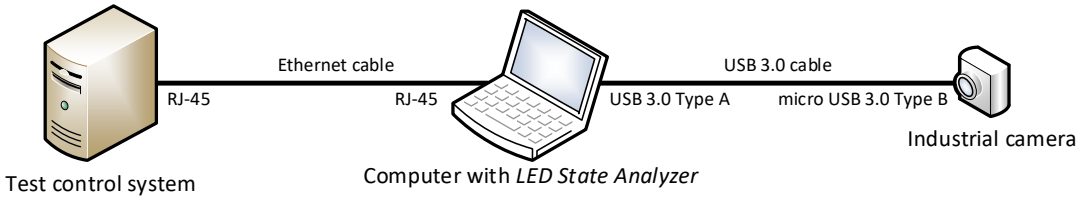


Figure 3.11: Wiring diagram of the system.

3.1.6 Target Platform

The target platform for *LED State Analyzer* is 64-bit Linux. The application was developed and tested on the 64-bit Linux distribution *Mint 18*, and this is also the only platform on which it was tested. The main reason for choosing Linux is to provide support for replacing a computer with Raspberry Pi or other embedded systems. However, this variant is out of the scope of this thesis.

3.2 Used Software Tools

LED State Analyzer is implemented in C++ by using third-party open-source libraries described in this section. Programming language C++ was chosen for its performance and support of used libraries. All used libraries and code samples are free to use for commercial applications.

3.2.1 Graphical User Interface (GUI)

For a user's comfort, *LED State Analyzer* uses GUI. There are several GUI libraries for C++, and it was necessary to select library which can be used for commercial purposes. One of these libraries is *GTK+* which is used here. This library is written in C, but there exists a C++ interface called *Gtkmm*. *Gtkmm* added support for using the library in an object-oriented C++ application. Used version of this interface is *Gtkmm 3.0*, which was the latest released version at the time of writing this thesis. [5].

There are two ways of a GUI design. It can be created directly in the source code by placing widgets to appropriate layouts. An advantage of this approach is that a programmer has the design of the application fully under control and the source code can be easily optimized. The second option is to use an application where GUI can be designed graphically and based on that design the source files (which can be loaded by a developed application) are generated. One of these applications for *GTK+* is *Glade*. *Glade* provides tools for design of complete applications GUI which is then saved as `.glade` file. *Gtkmm 3.0* provides tools for generating GUI using this file.

3.2.2 Industrial Camera API

Software package *pylon 5* provided by the company Basler is used for communication with an industrial camera. This package contains camera drivers for Linux, software for testing the camera – *pylon Viewer*, and C++ API for controlling a camera. There is also available very detailed *C++ Programmer's Guide for Linux* for the API and code samples of typical applications [6].

3.2.3 Modbus Communication Protocol

As it was already mentioned in Section 3.1.1, communication between *LED State Analyzer*, and the test control system uses Modbus protocol. Modbus is a protocol on the application layer which can exploit various network layers, data link layers, and physical layers.

Modbus is a client/server based communication protocol used for communication between devices connected to different types of buses or networks. A typical transaction between client and server without any error is shown in Fig. 3.12. The client sends initial request to the server, the server receives a request, performs required action and sends response data back to the client that processes them [7].

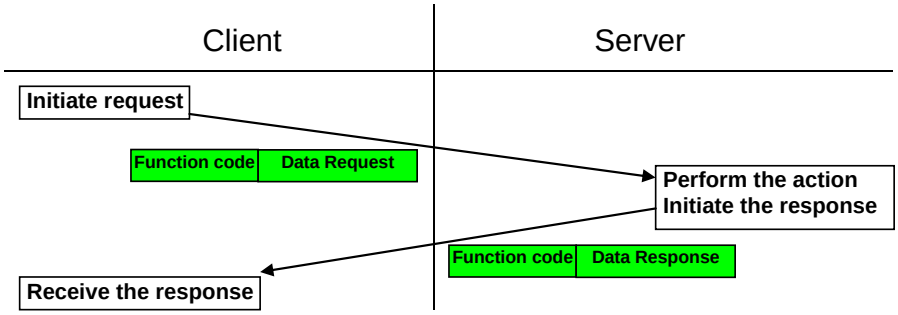


Figure 3.12: Modbus transaction [7].

Modbus data model is based on a series of tables [7]. Four primary tables are described in Tab. 3.2.

Table 3.2: Four primary tables of Modbus data model [7].

Primary tables	Object type	Type of
Discretes Input	Single bit	Read-Only
Coils	Single bit	Read-Write
Input Registers	16-bit word	Read-Only
Holding Registers	16-bit word	Read-Write

Library *libmodbus*

Implementation of Modbus communication interface is out of the scope of this thesis and therefore one of the already existing solutions is used, i.e., the library *libmodbus* – an open-source library written in C [8].

LED State Analyzer uses *libmodbus 3.0.6* which was the latest released stable version at the time of writing this thesis. An interface on a client side (test control system) uses *libmodbus 1.2.0* which is one of the first versions of this library. This version does not support all features as the newest version, but it is preferred by our customer. Both versions of the library implement the same communication protocol, and therefore different versions can be used on a client and server side.

3.2.4 Storing of Configuration and Data Files

For its correct functionality, *LED State Analyzer* requires many configured parameters. Values of these parameters can be changed by a user and saved for future use. To implement this functionality, it is necessary to choose a suitable format for storing these data. Based on the previous experiences we decided to use *JSON*. *JSON* is language independent data format, which can store various types of data (strings, numbers, arrays, etc.) [9]. Structure of data in *JSON* is also easy to understand and data files can be easily edited by a user. Based on these properties, *JSON* seems to be suitable for this kind of application.

Library *JSON for Modern C++*

For creating new JSON files and parsing already existing files, it is necessary to use interface which provides these functionalities. For that purpose, some of the already existing libraries can be used. In *LED State Analyzer*, the library *JSON for Modern C++* was used [10]. This library provides all required functionalities and is easy to use. Used version of this library is *JSON for Modern C++ 3.0.1*.

3.2.5 Tools for Digital Image Processing

The significant part of this thesis deals with digital image processing. Most of the used algorithms are very common in image processing and therefore many algorithms implementations already exist. There is no need to implement these algorithms again, therefore, *OpenCV* library is used.

Library *OpenCV*

OpenCV (Open Source Computer Vision Library) is one of the biggest libraries providing tools for many different operations related to the digital image processing. This library is an open-source and supports several programming languages including C++ [11]. *LED State Analyzer* uses *OpenCV 3.4.0*.

3.3 Digital Image Processing

Digital image processing is a processing of images using a computer. The main goal of image processing is to extract meaningful information from an image by performing appropriate operations. For the purpose of this thesis, the meaningful information are states of LEDs of observed devices, and this information is extracted from captured image. All methods of image processing used in *LED State Analyzer* are described in this section. [12, 13]

3.3.1 Analysis Process

A diagram illustrating the whole process of image analysis implemented in *LED State Analyzer* is shown in Fig. 3.13.

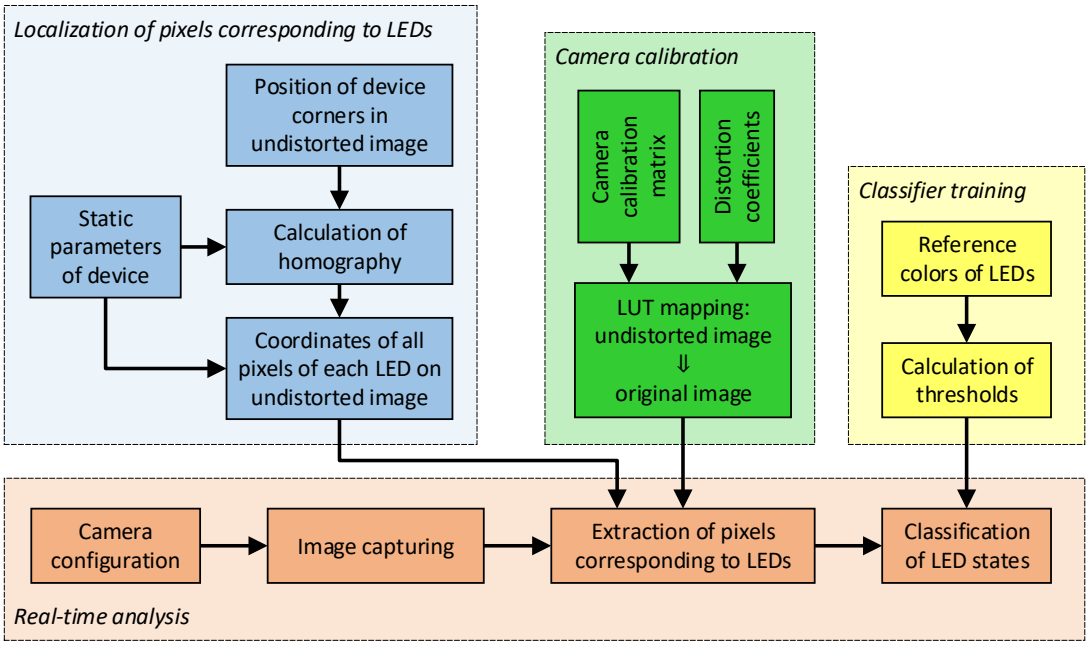


Figure 3.13: Diagram of the analysis process.

Image processing requires a lot of processor performance, and therefore it is necessary to optimize the analysis process. This process has four parts: localization of pixels corresponding to LEDs, camera calibration, classifier training, and real-time analysis. First three parts are performed only during the application configuration but the real-time analysis is performed during each recognition of LEDs states, and therefore it is much more time critical. The primary goal of optimization was to minimize total time duration of all performed actions executed during real-time analysis. Each part of the analysis process is described in more details below.

Localization of Pixels Corresponding to LEDs

Coordinates of pixels corresponding to LEDs are calculated based on known coordinates of device corners on an undistorted image and its dimensions (see Appendix A). This part of the analysis process is performed whenever a position of any device corner is changed.

Coordinates of device corners can be found automatically by using methods for object recognition (e.g., pattern recognition or neural networks [14]), or manually by a user. Automatic object recognition typically requires good object description, and therefore it would be complicated to add support for a new type of device. Automatic object recognition is also much less reliable under different lighting conditions. Because of these disadvantages, localization of device corners is done manually by a user. Before starting the analysis, a user is prompted to mark all corners of all devices on an undistorted image, and these coordinates are used for following calculations.

The application also includes a possibility to correct coordinates of marked corners when the test setup stays unchanged, but a camera was moved. In this case, a user can run an object tracking which should correct marks of corners based on the current

image. The principle of object tracking is described in Section 3.3.4.

Based on the device size (width and height) and the coordinates of device corners on an undistorted image, homography between the position of the device corners on the undistorted image (in pixels) and the real dimensions of the device (in millimeters) is calculated. The process of finding homography is described in Section 3.3.3.

Coordinates of all pixels corresponding to each LED can be calculated by using inverted homography and known positions and dimensions of LEDs in millimeters (device description). This process generates a list of coordinates of all pixels corresponding to each LED of a device.

Camera Calibration

LUT mapping is used to map an undistorted image to the original image – obtaining coordinates of a pixel on the original image when the coordinates of this pixel on the undistorted image are known. LUT mapping has the same dimensions as the captured images (in pixels) and a value of their every element corresponds to coordinates of a pixel on the original image.

LUT mapping is calculated based on user setting of camera calibration matrix and distortion coefficients. The LUT is updated whenever a camera calibration matrix or distortion coefficients are changed. The process of getting distortion coefficients, camera calibration matrix, and calculation of LUT mapping is described in Section 3.3.2.

Classifier Training

Training of classifier for classification of LED state is performed according to reference colors of LEDs. These colors are used for calculations of linear classifiers between different LED states. Classifier training is performed every time when a reference color of any LED is changed. Classification is described in Section 3.3.5.

Real-Time Analysis

Real-time analysis is a time-critical part of an analysis process, and therefore it is necessary to minimize total time duration of all performed actions. The whole process of real-time analysis is performed whenever an actual state of any LED is requested.

The first step is a configuration of the camera. Configurable parameters include global gain, exposure time, and white balance. After successful configuration, a new image is captured and converted to an appropriate format. Then pixels corresponding to LEDs can be extracted from this image. Pixels extraction is performed by using known coordinates of pixels corresponds to LEDs on an undistorted image and mapping LUT from undistorted image to original image. LED states classification follows when pixels of each LED are known. Classification process is described in a detail in Section 3.3.5.

3.3.2 Camera Calibration

Camera calibration is a process of estimating camera parameters using images of a known pattern captured from different angles by this camera. For modeling of a real camera with a lens, linear pinhole model cannot be used (see Section 3.3.2). Image projection of a real camera is nonlinear, and therefore it is necessary to consider projection deviations. The most significant impact on the camera projection error has the lens distortion. Results of a camera calibration are coefficients of lens distortions and camera calibration matrix. When these parameters are known, a LUT mapping from an undistorted image to a distorted image can be calculated [15].

Pinhole Camera Model

The simplest model of the camera is a pinhole model. This model considers a camera without any lens and with an aperture as a single point (small hole). Geometric illustration of a pinhole model is shown in Fig. 3.14. In the illustration, there is a projection of a point $\mathbf{X} = (X, Y, Z)$ to the image plane in the distance f from the camera center C. Intersection of the image plane and principal axis is in the point p [15, 16].

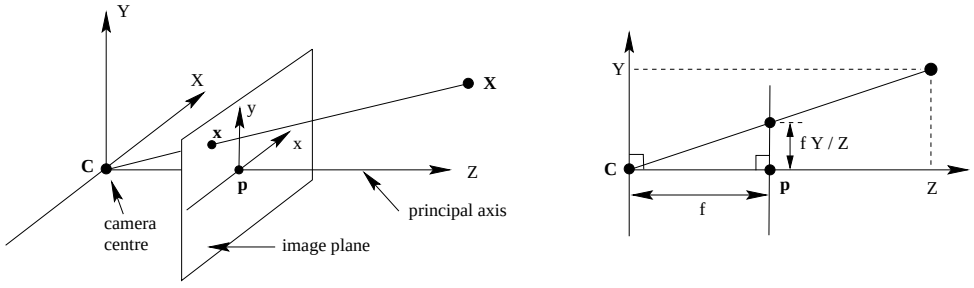


Figure 3.14: Geometric illustration of a pinhole model [16].

Central projection mapping from world to image coordinates is given by Eq. 3.3. An image has only two coordinates – x and y because z coordinate is constant for all mapping and it is always equal to focal length f [16].

$$(X, Y, Z)^T \mapsto \left(f\frac{X}{Z}, f\frac{Y}{Z} \right)^T \tag{3.3}$$

Using pinhole in a real world is not a proper way to capture images because a hole has to be as small as possible (theoretically infinitely small), and therefore there is not enough light on the image plane for a short exposure [15].

Lens Distortions

The main reason for estimation of camera parameters is to use them for correction of lens distortions. There are two considered types of lens distortion: radial distortion and tangential distortion. Lens distortions can be described by a Taylor expansion of even function. The accuracy of the description of distortions is given by the number of

used parameters of this expansion. Typically, it is sufficient to use only a few first of them [15, 16].

Radial distortion is usually the most significant one. It is caused by the optical properties of a camera lens. Distortion is zero in the center of the imager, and it increases as we approach its edges [15]. With decreasing price and focal length of the lens, this error becomes more significant [16]. There are two types of radial distortion – negative radial distortion ("pincushion") and positive radial distortion ("barrel"). Comparison of both of these radial distortion types and an undistorted image is shown in Fig. 3.15.

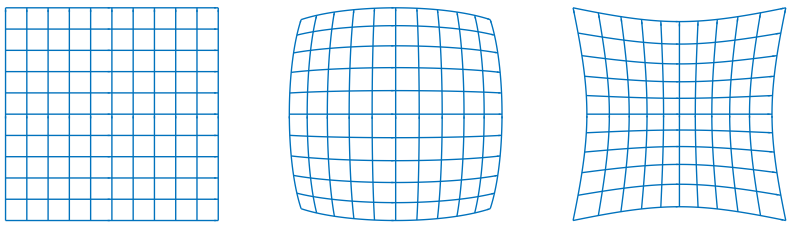


Figure 3.15: Illustrations of radial distortion. From the left: no distortion, positive radial distortion, negative radial distortion.

Radial distortion is typically described by three coefficients k_1, k_2, and k_3. When a distortion is rather small, it is sufficient to use only first two coefficients. With known coefficients of a radial distortion, a corrected location of each pixel can be calculated based on its known location on distorted image by using Eq. 3.4, 3.5, and 3.6, where $(\hat{x}, \hat{y})$ are corrected coordinates, (x, y) are coordinates on distorted image and (x_c, y_c) is the center of distortion [16].

$$r^2 = (x - x_c)^2 + (y - y_c)^2 \tag{3.4}$$

$$\hat{x} = x_c + (x - x_c)(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.5}$$

$$\hat{y} = y_c + (y - y_c)(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.6}$$

Second common type of distortion is tangential distortion. It is caused by manufacturing defects due to which a lens is not exactly parallel to the imager. Comparison of an image with tangential distortion and an undistorted image is shown in Fig. 3.16.

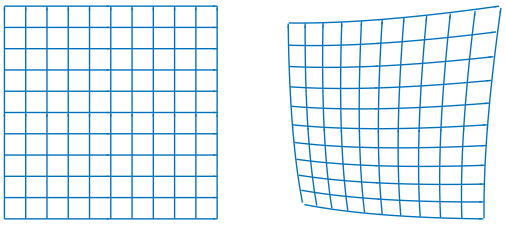


Figure 3.16: Illustrations of tangential distortion. From the left: no distortion, tangential distortion.

Tangential distortion is typically described by two coefficients p_1 and p_2. With known coefficients of a tangential distortion can be calculated corrected location of each pixel based on its known location on distorted image by using Eq. 3.4, 3.7, and 3.8, where $(\hat{x}, \hat{y})$ are corrected coordinates, (x, y) are coordinates on distorted image and (x_c, y_c) is the center of distortion [15].

$$\hat{x} = x_c + (x - x_c) + (2p_1(x - x_c)(y - y_c) + p_2(r^2 + 2(x - x_c)^2)) \quad (3.7)$$

$$\hat{y} = y_c + (y - y_c) + (p_1(r^2 + 2(y - y_c)^2) + 2p_2(x - x_c)(y - y_c)) \quad (3.8)$$

Both types of distortion can be corrected by merging of equations for correction of radial distortion and tangential distortion.

Camera Calibration Matrix

Camera calibration matrix K is used for transformation from the world coordinates to the image coordinates. K is a 3×3 matrix defined by Eq. 3.9, where f is a camera focal length and (p_x, p_y) are the coorindates of a principle point (see Fig. 3.14) [16].

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.9)$$

Calculation of LUT Mapping

LUT mapping to map pixels of an undistorted image to the pixels of the original image can be calculated by using camera calibration matrix and distortion coefficients. LUT mapping gives for each pixel of an undistorted image, coordinates of the corresponding pixel on the original image. To obtain this LUT, the following procedure has to be performed for each pixel of an undistorted image:

1. let be (u, v) coordinates of an undistorted image,
2. transform (u, v) from the image coordinates to the world coordinates: $\begin{bmatrix} x & y & 1 \end{bmatrix}^T = K^{-1} \begin{bmatrix} u & v & 1 \end{bmatrix}^T = \begin{bmatrix} \frac{u-p_x}{f} & \frac{u-p_y}{f} & 1 \end{bmatrix}^T$,
3. apply equations for correction of distortions (Eq. 3.5, 3.6, 3.7, and 3.8) to the coordinates $\begin{bmatrix} x & y & 1 \end{bmatrix}^T$ to get corrected coordinates $\begin{bmatrix} \hat{x} & \hat{y} & 1 \end{bmatrix}^T$
4. transform $(\hat{x}, \hat{y})$ from the world coordinates back to the image coordinates: $\begin{bmatrix} \hat{u} & \hat{v} & 1 \end{bmatrix}^T = K \begin{bmatrix} \hat{x} & \hat{y} & 1 \end{bmatrix}^T = \begin{bmatrix} f\hat{x} + p_x & f\hat{y} + p_y & 1 \end{bmatrix}^T$,
5. save coordinates $(\hat{u}, \hat{v})$ to the element of LUT mapping which corresponds to the coordinates (u, v).

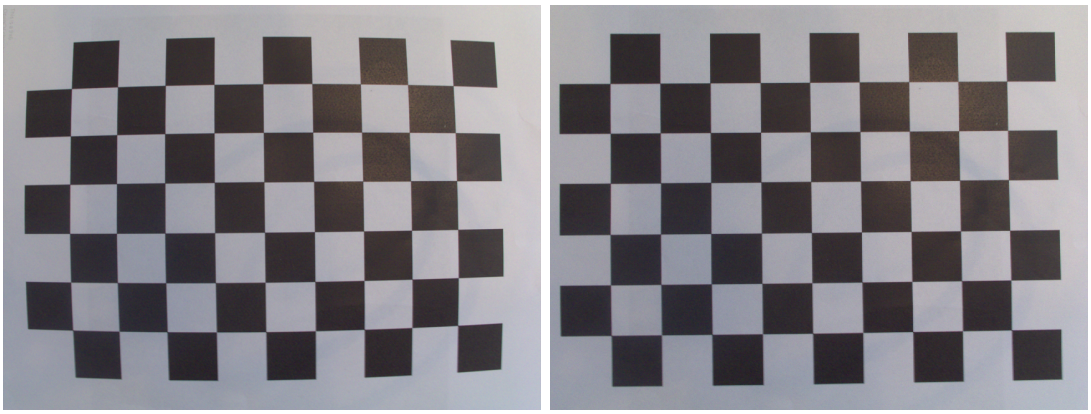
Calibration Procedure

As mentioned above, camera calibration matrix and distortion coefficients can be estimated by performing camera calibration. These parameters are typically estimated by using methods which minimize an error of reprojection using estimated parameters. An input of the process of calibration is a set of known point correspondences. These values are known if used images contain some pattern or object with known dimensions. In this thesis, a chess pattern printed on a paper was used. Estimation of camera parameters was performed by using function `calibrateCamera` from the library *OpenCV*. This library also provides a code sample for recognition of corners of a chess pattern in the image [17, 15].

The used camera was calibrated using seven images of a chess pattern printed on A4. The images were captured from different angles and camera positions. After processing these images by the function `calibrateCamera`, the following camera parameters were estimated:

$$K = \begin{bmatrix} 2800.52 & 0 & 1296 \\ 0 & 2800.52 & 972 \\ 0 & 0 & 1 \end{bmatrix}, \quad \begin{matrix} k_1 = -0.242 \\ k_2 = 0.033 \\ k_3 = 0.173 \end{matrix}, \quad \begin{matrix} p_1 = 0 \\ p_2 = 0 \end{matrix}.$$

These values correspond to the used camera *Basler daA2500-14uc* with lens *Basler Lens C125-0618-5M F1.8 f6mm*. The mean error of reprojection using estimated parameters is 1.3 pixels. This deviation is small enough to allow us to consider the image capturing as a linear projection (after correction by LUT mapping). Fig. 3.17 shows one of the images used for calibration before and after correction.



(a) Original distorted image. **(b)** Undistorted image (after correction).

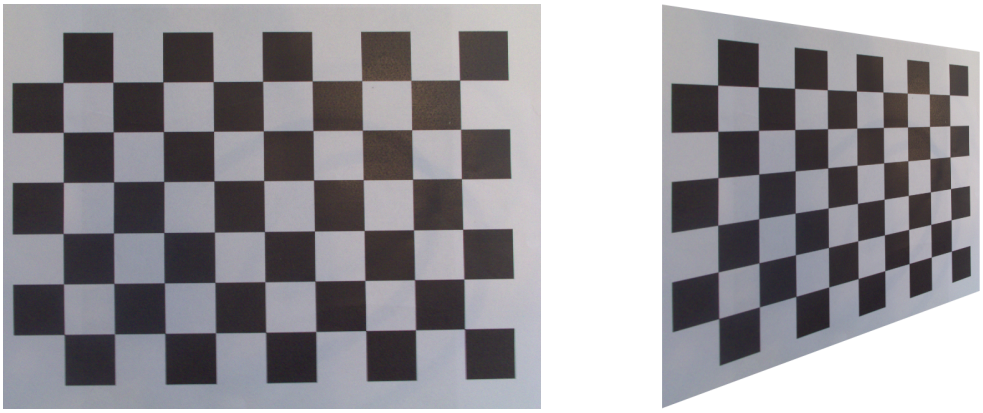
Figure 3.17: Comparison of an image captured for camera calibration before (3.17a) and after (3.17b) correction.

3.3.3 Finding Homography

A homography is any mapping $\mathcal{P}^d \rightarrow \mathcal{P}^d$ that is linear in the embedded space $\mathcal{R}^{d+1}$. It is given by Eq. 3.10, where H is a matrix with dimensions $(d+1) \times (d+1)$, which transforms point $\mathbf{u}$ in a space $\mathcal{P}^d$ to the point $\mathbf{u}'$ in another (or the same) space $\mathcal{P}^d$ [14].

$$\mathbf{u}' \simeq H\mathbf{u} \tag{3.10}$$

In this thesis, it is used only 2D homography and therefore $d = 2$. The 2D homography maps each point of one image to some point of another image using matrix H. Illustration of 2D homography applied to the image is shown in Fig. 3.18.



(a) Image before projective transformation. **(b)** Image after projective transformation.

Figure 3.18: Illustration of projective transformation. Fig. 3.18b was created by applying homography to the Fig. 3.18a.

For the case of 2D homography, Eq. 3.10 can be rewritten to the Eq. 3.11. This equation shows mapping of non-homogenous point (u, v) to the non-homogenous point (u', v'), where $\alpha \in \mathbb{R}\backslash\{0\}$ is a scale of (u', v'). α must be different from zero, otherwise, a projection of point (u, v) would be in infinity, which is a case we do not assume [14].

$$\alpha \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.11}$$

Coordinates of point (u', v') can be calculated by using Eq. 3.12 and 3.13. These equations are based on Eq. 3.11 with elimination of a scale α [14].

$$u' = \frac{h_{11}u + h_{12}v + h_{13}}{h_{31}u + h_{32}v + h_{33}} \tag{3.12}$$

$$v' = \frac{h_{21}u + h_{22}v + h_{23}}{h_{31}u + h_{32}v + h_{33}} \tag{3.13}$$

Calculation of the Homography Matrix

The calculation of the homography matrix will be based on Eq. 3.11 defined in a previous section. We are looking for a non-homogenous solution of this equation. Matrix H has nine elements but only eight unknowns, so we have to fix one of them. Fixing means setting one of the elements to any non-zero constant. We will set $h_{33} = 1$, which leads to Eq. 3.14 and 3.15.

$$u' = \frac{h_{11}u + h_{12}v + h_{13}}{h_{31}u + h_{32}v + 1} \tag{3.14}$$

$$v' = \frac{h_{21}u + h_{22}v + h_{23}}{h_{31}u + h_{32}v + 1} \tag{3.15}$$

After multiplying these equations by their denominators and expressing u' and v', we will get Eq. 3.16 and 3.17:

$$u' = h_{11}u + h_{12}v + h_{13} - h_{31}uu' - h_{32}vu' \tag{3.16}$$

,

$$v' = h_{21}u + h_{22}v + h_{23} - h_{31}uv' - h_{32}vv' \tag{3.17}$$

.

We can see that there are two equations for each corresponding pair of points (coordinates before and after transformation), and therefore to solve eight unknowns we need to know coordinates of four corresponding points. Based on this result we can construct matrix equation 3.18. By solving this equation, we will get the remaining coefficients of matrix H. This system of linear equations can also be solved for more than four known points. In that case, we will get an overdetermined system of linear equations which can be solved by using least squares method.

$$\begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -u_1u'_1 & -v_1u'_1 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -u_1v'_1 & -v_1v'_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u_2u'_2 & -v_2u'_2 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -u_2v'_2 & -v_2v'_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u_3u'_3 & -v_3u'_3 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -u_3v'_3 & -v_3v'_3 \\ u_4 & v_4 & 1 & 0 & 0 & 0 & -u_4u'_4 & -v_4u'_4 \\ 0 & 0 & 0 & u_4 & v_4 & 1 & -u_4v'_4 & -v_4v'_4 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} u'_1 \\ v'_1 \\ u'_2 \\ v'_2 \\ u'_3 \\ v'_3 \\ u'_4 \\ v'_4 \end{bmatrix} \tag{3.18}$$

In this thesis, there are two cases when the homography is calculated. The first case is performing of object tracking. The principle of object tracking is described in Section 3.3.4, but its primary goal is to customize the position of the devices' corners marked by the user according to the current camera position. The second case is finding homography between device on the image and its real dimensions. A user is responsible for localization of devices' corners on the image, and then it is necessary to find a homography between a device (described by its dimensions) and its corners marked on the image. Known coordinates of its four corners are enough for finding homography, which can be later used for localization of LEDs on the image based on their known position on a device.

3.3.4 Object Tracking

Object tracking is a process of finding homography between two images of the same object, while the object or camera was moved. In this thesis, it is used for localization

of devices that were already localized by a user, but after that, a camera was moved slightly. A typical example is a situation when the application is configured and devices are localized, and then the arm with the camera flips out. Later, when another test with the same configuration is going to be performed a camera is usually not in the exact same position as during the localization of devices. By using an object tracking, the configuration can be adapted to the actual camera position.

The whole process of the object tracking between two images can be done by performing following actions [18]:

1. detect and describe features of both images,
2. match keypoints across the images,
3. estimate a homography between the images.

Feature Detection and Description

Feature detection is a process of extracting points of interest (keypoints) from the image. These points are for example edges, corners or sharp changes of colors. Some of the commonly used algorithms for feature detection are BRISK, SIFT, SURF, ORB, KAZE, or A-KAZE. Essential parameters for choosing the most suitable algorithm are the time of detection and description of one feature and number of correct matches [14, 19, 20].

This thesis does not compare the algorithms for feature detection and matching, and, therefore, a choice of the most suitable algorithm is based on published papers dealing with comparisons of them. The considered algorithms are BRISK, SIFT, SURF, ORB, KAZE, and A-KAZE. The best choice seems to be A-KAZE (Accelerated-KAZE) introduced in 2013. It is faster than SIFT, SURF, and KAZE, and it is also more accurate than ORB and BRISK. Based on this properties, it was used in this thesis [19, 20, 21].

When the keypoints of both images are known, their descriptors can be calculated. A descriptor of a keypoint is created based on color intensity changes in the near area around the keypoint. It is represented by an array of binary values which corresponds to change of intensity between a pair of pixels: 0 when intensity decrease, 1 when intensity increase.

Keypoints Matching

Matching of keypoints is a process of assigning the corresponding keypoints between the images according to their descriptors. We used the brute force method in this thesis. For each keypoint on the first image, the corresponding keypoint on the second image is found. The degree of similarity between two keypoints is given by Hamming distance of their descriptors. Because descriptors are in the form of a binary array, Hamming distance seems to be an appropriate choice [18].

The keypoints across the images could be matched using 1-nearest neighbor classifier. A principle of this method is to match each keypoint of the first image with its nearest neighbor in the second image (using a Hamming distance of their descriptors).

This approach is possible but it can be improved by using a ratio of distances between two nearest neighbors. In this method, the 2-NN classifier is used and for each keypoint it is calculated if the distance to its nearest neighbor is smaller than the distance to its second nearest neighbor multiplicated by a threshold distance ratio. If it is so, keypoints are added to the list of matched keypoints, and the algorithm continues with the next keypoint of the first image. Otherwise, this keypoint is not used [22].

Fig. 3.19 shows an example of probability density functions for correct and incorrect matches in terms of the ratio of nearest to second nearest neighbors of each keypoint. Based on this figure, the threshold of a distance ratio of 0.8 was chosen. This means that the keypoint from the first image is matched with its nearest neighbor from the second image only if the distance ratio between the nearest neighbor and the second nearest neighbor is less or equal to 0.8.

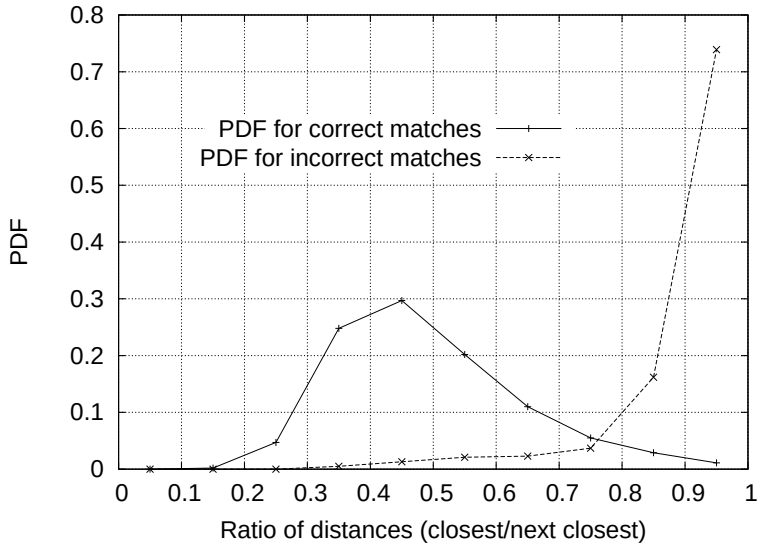


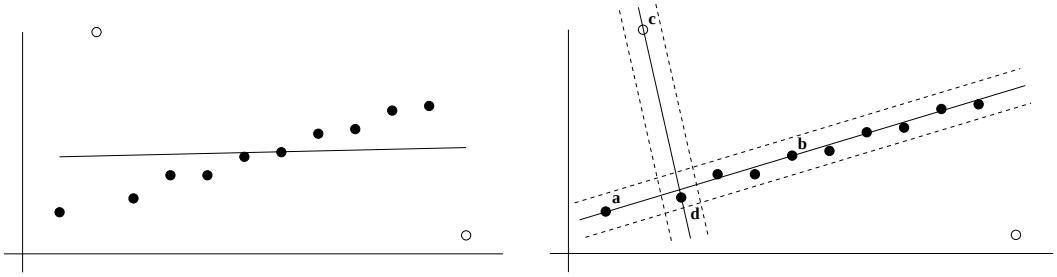
Figure 3.19: Example of probability density functions for correct and incorrect matches in terms of the ratio of nearest to second nearest neighbor of each keypoint [22].

Estimation of Homography

When the list of corresponding keypoints is complete, a homography between images can be calculated. Although most of the keypoints in the list should correspond (inliers), there can also be some keypoints which are not corresponding (outliers). Before calculation of the homography, we should distinguish between these two groups. Otherwise, a few poorly assigned keypoints could significantly decrease the accuracy of the homography. It follows that a robust method for estimation of the homography should be used.

For estimate of the homography, the method called RANSAC (Random sample consensus) was used. RANSAC is an iterative method for estimation of parameters (in this case homography) from a set of data containing outliers. Unlike the least-squares method, this method excludes outliers, and therefore estimated parameters are typically

much more accurate. Comparison of these two methods is shown in Fig. 3.20 [16].



(a) Estimation using least-squares method. **(b)** Estimation using RANSAC method.

Figure 3.20: Estimation of parameters of linear function from set of points by using least-squares method (3.20a) and RANSAC method (3.20b) [16].

This figure shows an estimation of a linear function from the points. There are ten inliers and two outliers. It can be seen that estimation by least-squares method is less accurate than by RANSAC method. The accuracy of estimation by using RANSAC method is measured by the number of points within a threshold distance of the line (in Fig. 3.20 thresholds are dotted lines). It can be seen, that when a line is estimated using points **a** and **b** (both inliers), all inliers are within threshold distances (10 points are between thresholds). But when a line is estimate using points **c** and **d** (only **d** is inlier), only one inlier is within threshold distances (2 points are between thresholds). Line estimated using points **a** and **b** is evaluated as more precise because there are more points within threshold distances ($10 > 2$)[16].

The input of a RANSAC method used in this thesis is a set of matched keypoints. RANSAC threshold (R_{thr}) is set by a user. The goal of an algorithm is to estimate homography between two images. Homography is estimated by performing following steps:

1. select a subset of matched keypoints randomly,
2. calculate homography for this subset,
3. apply homography H to all matched keypoints of the first image x_i and compare the result with a position of the corresponding keypoint on the second image y_i,
4. split keypoints to inliers and outliers
   (a) inlier: $||y_i - H \cdot x_i|| \leq R_{thr}$
   (b) outlier: $||y_i - H \cdot x_i|| > R_{thr}$,
5. save number of inliers and outliers for this homography,
6. if the algorithm was performed N times exit it, otherwise, continue from the step 1.

After exit of the algorithm, the homography with the most inliers is chosen as the most accurate and it is used for subsequent actions [17, 15].

3.3.5 LED State Classification

Classification of LED state is possible when colors of all pixels corresponding to the LED are known. These pixels can be extracted from the image by using known dimensions of observed device and homography between a device and an image as described in Section 3.3.1. The color of each pixel is characterized by three color channels: red (R), green (G), and blue (B).

Classification is optimized for recognition of two-colored SMD LED, which consists of red and green LED, with a diffuser. There are four possible states of the LED and all of them are described in Tab. 3.3. Based on this consideration, we can assume that only red and green color channel is important for LED state recognition. This assumption was checked by multiple measurements of colors of LEDs in different states and comparison of intensities of the blue color channel from these measurements. Based on this test, any correlation between the intensity of the blue channel and the state of the LED was not determined. Therefore, only the intensity of red and green color channel for classification is used.

Table 3.3: Considered states of LED.

Green LED	Red LED	LED state (observed color)
off	off	off
on	off	green
off	on	red
on	on	orange

An important step before classification of LED state is a selection of pixels corresponding to it because classification of wrong pixels can increase classification error. There are two supported shapes of a LED: square and round. If the LED has a shape of a rectangle or a square, all pixels inside it are analyzed. If the LED has a shape of a circle, it is computationally more demanding to check if the pixel is inside the circle (especially after projective transformation of this circle). In that case, the LED is considered to be a square with a side three-quarter of a circle diameter. Illustration of this simplification is shown in Fig. 3.21.

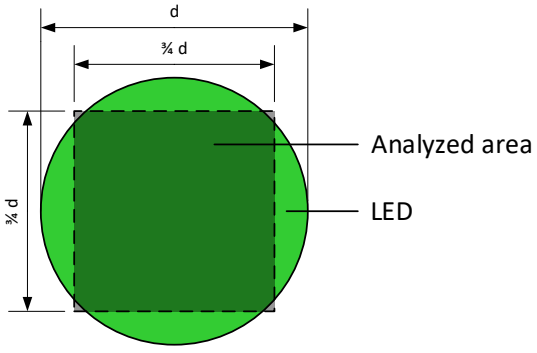


Figure 3.21: Analyzed area for state recognition of a round LED.

The number of pixels on an image corresponding to a LED depends on many factors, such as the size of LED, the distance between the camera and the observed device, etc. Theoretically, the number of pixels can be from one to the total number of pixels on the image, but a classification of LED state based on only a few pixels would be error-prone. It led to the introduction of restriction which allows classification only if each LED is visible on at least 50 pixels.

Since the number of pixels for classification is not known a classifier has to be independent on their count. It is also necessary to assume that some of the classified pixels do not correspond to LED (LED is not localized precisely) and if the number of these pixels is not too significant, they should not influence classification accuracy. The colors of pixels corresponding to randomly selected LEDs in different states is shown in Fig. 3.22. This figure shows pixels as points located based on their intensity of red and green channel. All the pixels were extracted by using the procedure described in Section 3.3.1. The most significant impact on the final result has a camera configuration (exposure time, gain, and white balance) and its location.

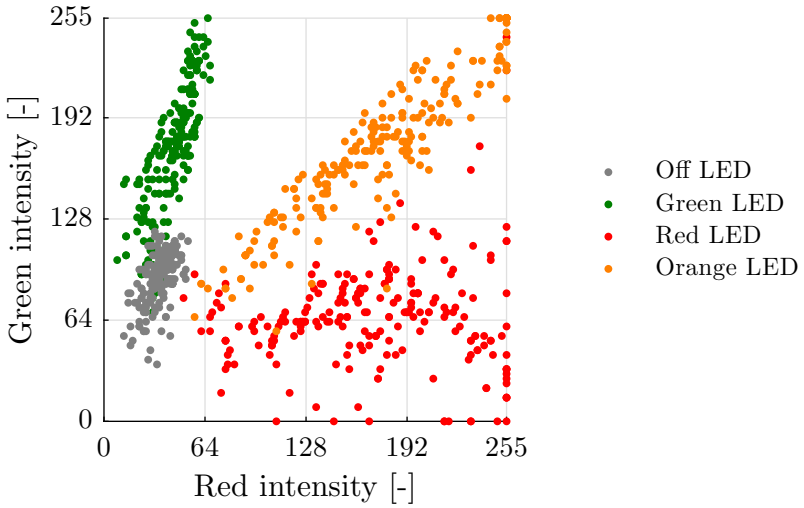


Figure 3.22: Color of pixels corresponding to each LED state (only red and green color channel).

It is evident that colors of pixels of LEDs form four clusters. Clusters are not sharply divided, but they are still easy to see. For a better illustration Fig. 3.23 shows average colors of the LEDs and their covariance ellipses.

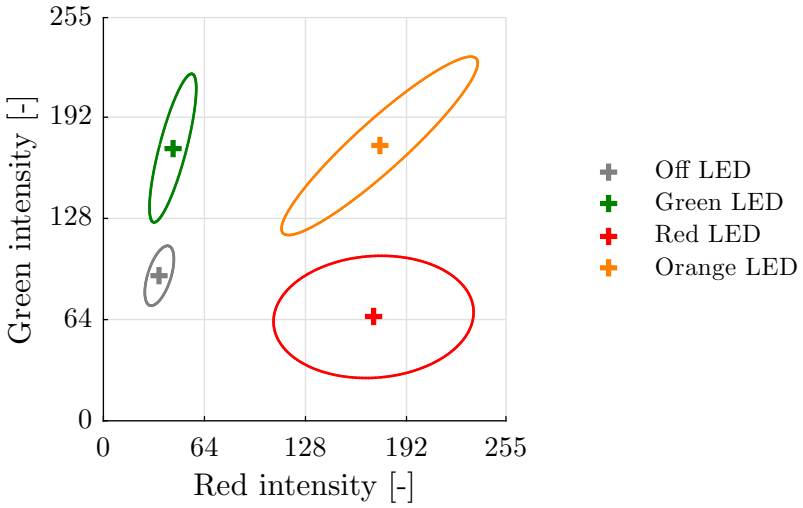


Figure 3.23: Average color of pixels corresponding to each LED state with covariance elipse around it (only red and green color channel).

It can be seen in the figure that spaces between average colors of different LEDs are relatively big and they should be readily separable. It is expected that, before classification, a user sets a reference color of each classified LED state. Based on these values thresholds between them can be calculated. These thresholds can be used for LED state classification.

The light intensity when the LED is turned on (green, red, and orange) can differ according to the relative position of camera and device. This fact was considered when finding the most suitable thresholds for classification. Covariance ellipses show that green LED expand mostly in the green channel, red LED in red channel and orange in both of them. Based on these results, suitable thresholds between green, red, and orange color could be set as a line passing through the origin and middle of the two neighboring reference colors. Threshold between off state and green state is set as a horizontal line passing the average green channel intensity of these two states. The threshold between off state and red state is set as a vertical line passing the average red channel intensity of these two states. The threshold between off state and orange state is set as the line between the intersection of the horizontal line and the threshold between the green and orange and the intersection of the vertical line and the threshold between the orange and the red. Model example with added thresholds is shown in Fig. 3.24.

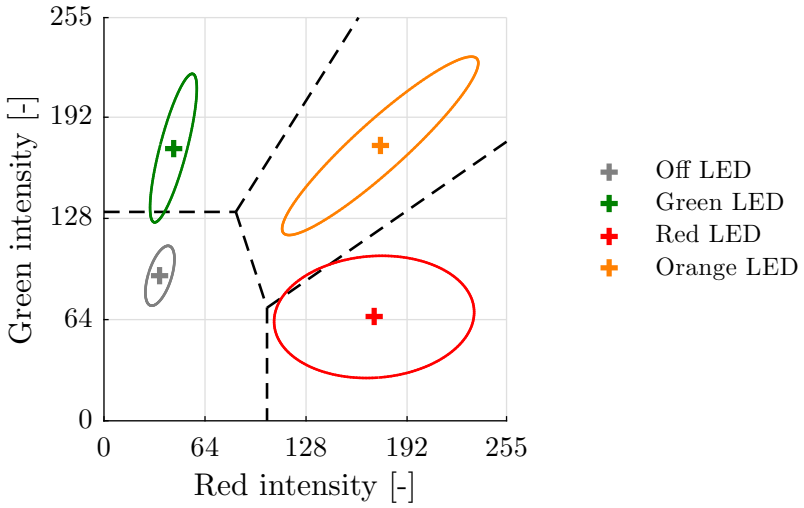


Figure 3.24: The average color of pixels corresponding to each LED state with covariance ellipse around it and thresholds between classification classes (only red and green color channel).

It can be seen in this figure, that even if the intensity of brightness of green, red or orange LED were higher, most of the pixels of LED would be still between the same thresholds and therefore it would be correctly classified. The same thing does not apply to the case when the intensity of brightness would be much lower than the light of reference LEDs. In that case, a LED with the lower intensity could be classified as off although it would be turned on. In this situation, the positions of thresholds can be changed manually by a user. Detailed information about this functionality is written in the user's manual in Section 4.6.1.

The LED classification is done by performing following steps:

1. select one pixel out of the set of pixels corresponding to LED,
2. compare its color to the threshold and classify it,
3. increase the number of pixels classified to corresponding class (n_{off}, n_{green}, n_{red}, or n_{orange}) by one,
4. if all pixels were already classified continue, otherwise, repeat from the step 1.,
5. evaluate most probable state of the LED based on the number of pixels classified to each class ($\max\{n_{off}, n_{green}, n_{red}, n_{orange}\}$).

This approach prevents significant impact on the classification by a few outliers. It can be useful when not all of the pixels belong to the LED, so these pixels do not have an impact on the classification if there is more than 50 % of correctly classified pixels. Classification map created according to the thresholds and colors of pixels of reference LEDs that was used for calculation of thresholds is shown in Fig. 3.25.

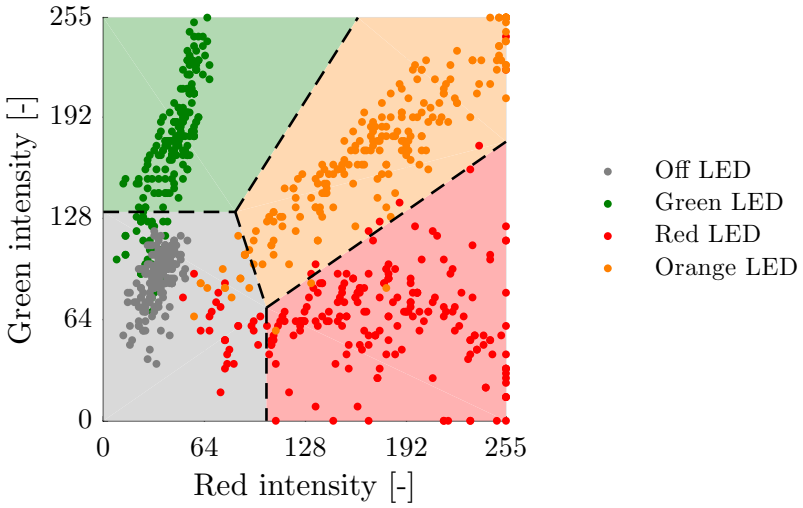


Figure 3.25: Example of classification map of randomly selected reference LEDs and all pixels corresponding to these LEDs.

It can be seen that most of the pixels are classified correctly, and only a few of them are in a wrong class. In particular, pixels belonging to LEDs which are turned on and are classified as off, may be classified actually correctly. These pixels may belong, for example, to the front panel instead of the LED and they are only erroneously extracted from the image.

This type of classifier is used for all classification of LEDs in *LED State Analyzer*. In the application, the probability of belonging to each class is also calculated. It is calculated as a percentage of all pixels classified to the particular class.

3.4 Application Design

LED State Analyzer is designed as an object-oriented application written in C++ using external libraries written in C++ or C. The whole program can be split into the four following blocks:

- devices database and LED state analysis,
- GUI and file management,
- camera interface,
- interface with the test control system.

Simplified class diagram with an illustration of these blocks and classes which belongs to them is shown in Fig. 3.26. This diagram also shows the use of external libraries by the classes (*Gtkmm* is used by all classes in the block). Names of classes are the same as in the application source code. A brief description of the classes follows:

- **MainWindow** controls the main window of *LED State Analyzer*. All handlers of GUI of this window are present in this class.

- **Preferences** controls window of preferences of camera and image processing. All handlers of GUI of this window are present in this class.
- **ClassifierSetup** controls window for editing parameters of classifier of a device. All handlers of GUI of this window are present in this class.
- **About** controls window with information about the application. All handlers of GUI of this window are present in this class.
- **DrawingLib** provides tools for drawing shapes used in the application GUI.
- **CameraController** implements interface between camera and the application. This class provides methods for configuration, controlling the camera, and converting captured image to the suitable format.
- **Device** describes one DUT whose LEDs are analyzed. This class contains information about device type, parameters, and classified states of LEDs.
- **Analyzer** provides tools for recognition of LEDs states of DUTs from the captured image. This class also provides tools for the correction of image distortions and object tracking.
- **ModbusServer** implements Modbus server used for communication with the test control system.

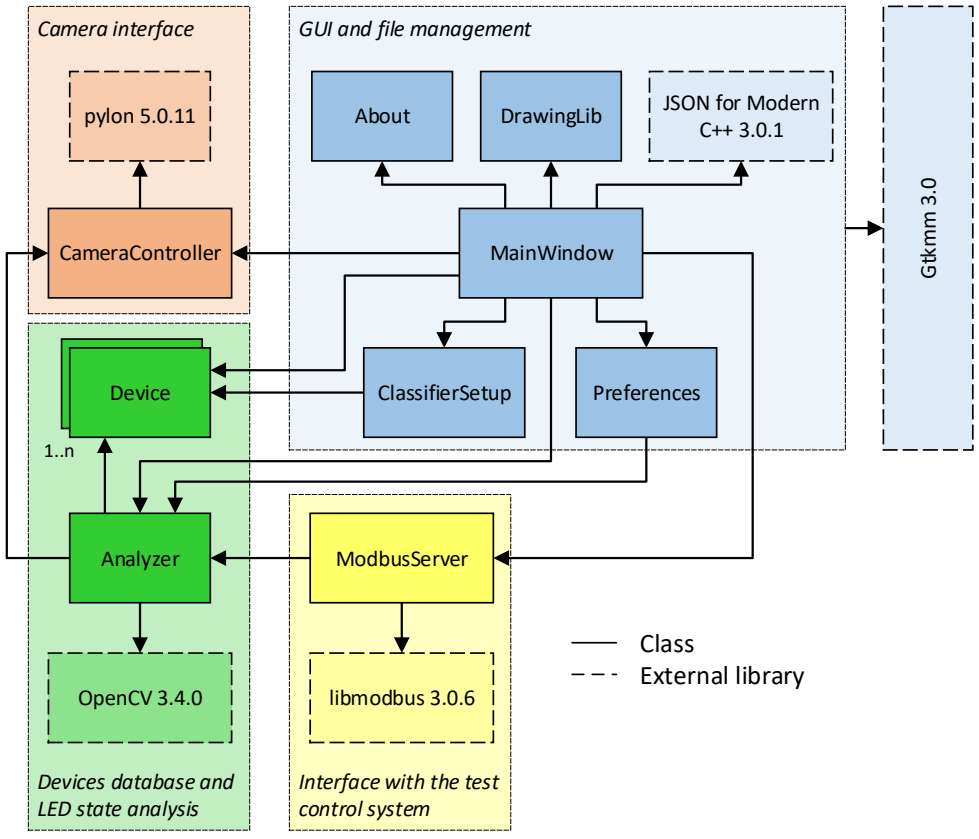


Figure 3.26: Simplified class diagram of *LED State Analyzer*.

3.4.1 Design of GUI

GUI of the whole application was designed using *Glade* (see Section 3.2.1). *Glade* generates XML file (filename extension is `.glade`) which can be loaded and used for creating GUI after the application launch. *LED State Analyzer* consists of four windows – the main application window, a window with preferences, a window for customization of LED states classification, and a window with information about the application. Each of them is controlled by separate class as it is shown in Fig. 3.26.

3.4.2 Camera Interface

An interface between the application and industrial camera uses library *pylon 5.0.11*. The interface was tested only with the camera *Basler dart daA2500-14uc*, and therefore it does not guarantee correct functionality with a different camera. Initialization of the instance of class `CameraController` requires a connected camera.

After successful initialization, it is possible to configure camera setting or to capture images. `CameraController` allows setting of exposure time, global gain and white balance. Other camera parameters are left at their default values. Function for taking an image sends a request to the camera and reads received data. For image processing, the *OpenCV* library which uses class `Mat` for image interpretation is used and therefore the received data are converted to an object of this class. Function for taking an image then returns an object of type `Mat` which represents a new image and is suitable for further processing.

3.4.3 Stored Data and Their Format

All data saved and loaded by the application are stored in *JSON* files. There are two types of files which contain information necessary for application configuration: `config_data.json` and `database_of_devices.json`. These two files are necessary for the launch of application or loading of configuration saved in the past. When the configuration is saved by the user, these two files are created. For the default configuration, both of these files are already in the `gobal_data` folder. Without these files, it is not possible to create a new project.

Test Setup Configuration

Configuration of the test setup is stored in files called `config_data.json`. These files contain following information: camera matrix, camera distortion coefficients, camera gain, white balance parameters, time of exposure for localization of devices' corners, time of exposure for LED state analysis, RANSAC threshold, IP address, and port number of a Modbus server and list of all devices with information about their type ID, name, corners positions and parameters of classification thresholds.

File `config_data.json` does not contain any information about static parameters of devices (shape, number of LEDs, etc.). This file only contains identification number

(ID) of the device type and this ID corresponds to some data record with its static parameters in file `database_of_devices.json`.

Static Parameters of Devices

Each type of device is described by its static parameters. These information is stored in the file called `database_of_devices.json`. In this file, there is an array whose every element contains static parameters of one device type. The element number in the array matches the device type ID. Parameters describing a device type are following: name of a device type, device width, device height, possible LED colors (green, red, orange, or arbitrary combinations of them), width of LED, height of LED, shape of LED (rectangle or circle), and an array of parameters describing blocks of LED. Each block contains information about horizontal and vertical coordinate of the top left LED, horizontal and vertical space between LEDs, number of LEDs in one column, and number of LEDs in one row. Meaning of the device dimensions is described in Appendix A.

3.4.4 Multithread Approach

After startup, the application is running in one main thread. The main thread is used for controlling GUI, performing configurations, and testing the LED state recognition accuracy. When the real-time analysis starts, a new thread is created. This thread is responsible for the creation and controlling a Modbus server and also for the LEDs states analysis when a request from the client is received. When the real-time analysis stops, the second thread is safely deleted and program continues to run only in the main thread.

The second thread is created by using POSIX threads. It has to be able to control GUI (update of communication log) which means that the GUI is controlled by both threads (by the main thread and this one). This situation could potentially cause application instability and therefore tools for multithread applications provided by *Gtkmm* are used.

3.4.5 Error Handling

For the correct functionality of the application, it is necessary to handle many situations which could lead to application instability or crash. These situations can be caused by a user or connected devices. Some of the most critical cases and their handling are written in the following paragraphs.

A user has to perform camera initialization before starting to use it. Another step of configuration is possible only after successful initialization of the camera. Otherwise, the user is not allowed to continue. A connection of the camera is also checked when the image is taken (manually by a user or during a real-time analysis). If there is a problem with communication with the camera, a user is informed that the camera was disconnected and for the next work it is necessary to connect and initialize it again.

When the previously saved test setup configuration and a device database are

loaded, consistency of these files is checked. If any file is corrupted or some data records are in a wrong format, the configuration is not used, and a user is informed about the possibility of files corruption.

When a user localizes devices' corners, it is checked if corners positions make sense. Each device has the shape of a rectangle, and therefore corners have to always make a convex polygon (independent on camera position). Convexity check can be performed by checking if all corners are oriented clockwise or counter-clockwise. If the orientation of any corner differs from others, then the polygon is not convex and a user is prompted to change a position of corners.

During localization of devices' corners by a user, it is checked whether areas which are expected to correspond to LEDs are sufficiently large for an analysis. The frame around each LED on a device must contain at least 50 pixels. This value is considered to be a threshold for accurate classification of LED state and it was found through an experiment. If there are fewer pixels inside LED frames, a user is prompted to change the location of corners.

3.4.6 Communication Between *LED State Analyzer* and the Test Control System

The communication between *LED State Analyzer* and the test control system was already mentioned in the Section 3.1.1. Illustration of the process of real-time analysis from the communication perspective is shown in Fig. 3.27. This diagram is split into two parts: initialization and cyclical requesting of LEDs states. The illustration also shows which actions are accomplished by a server and which of them are accomplished by a client.

The first step of the communication process is the start of Modbus server (*LED State Analyzer*) and its initialization. Client (test control system) can then connect to it and send a request for new LEDs states. When the server receives a request, the whole process of analysis described in Section 3.3.1 is performed. After completing the LEDs state recognition, new data are stored in Modbus registers. When the client receives them it can send next request. The next request shall not be sent before receiving the response to the previous one.

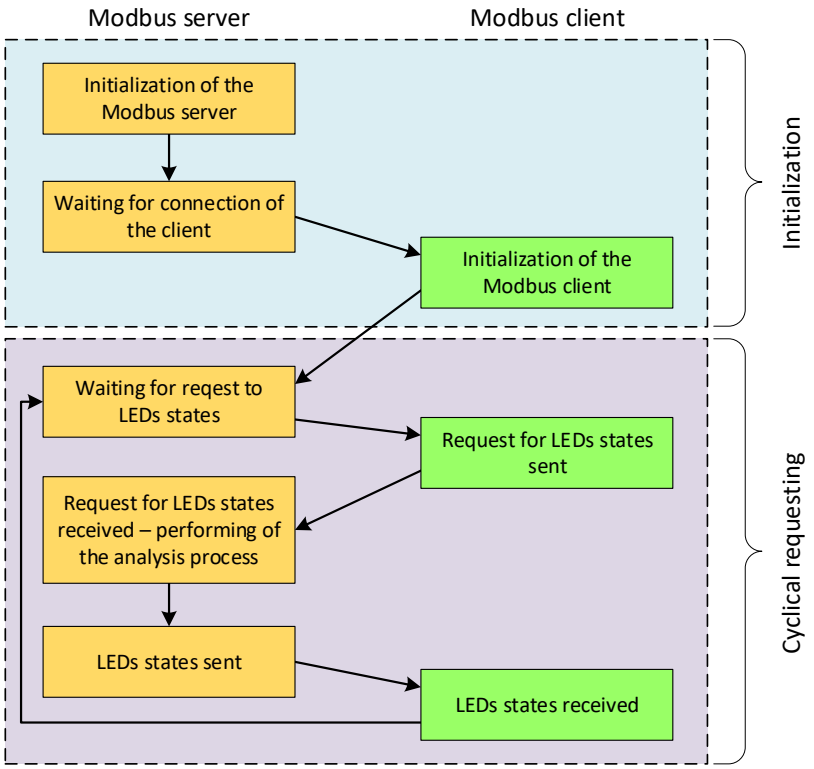


Figure 3.27: Process of a real-time analysis from the communication point of view.

A server can be closed only by a user or due to an unexpected error. When client closes the connection on its side, the server is waiting until client restores the communication or the user closes the server. This property is implemented based on the requirements in Chapter 2.

Structure of a Data Frame

Data provided by *LED State Analyzer* are stored in 16-bit Modbus holding registers. After request from a client, current LEDs states are recognized and these data are converted to a sequence of bits which is saved to the registers. Structure of data in registers is shown in Fig. 3.28.

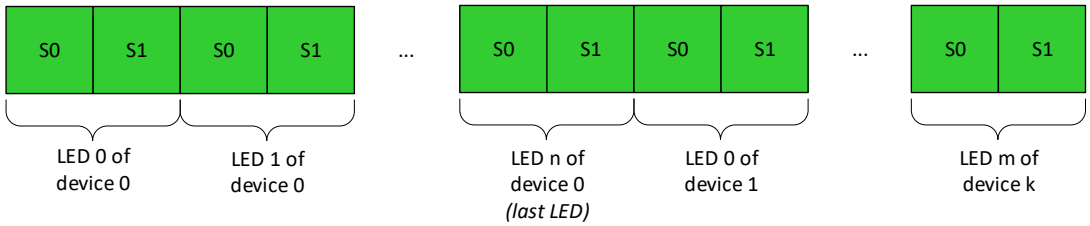


Figure 3.28: Illustration of data structure sent from a server to the client.

State of each LED of each device is characterized by two bits. Every Modbus register can hold states of up to eight LEDs. Data are stored in the following order: state of the first LED of a first device is stored in two LSB of the first (zero) register, a state

of the last LED of the last device is stored in the last used register. States of all LEDs of the first device from the lowest LED number to the highest are stored firstly. LEDs of the second device follows in the same order, etc. There is no space between LEDs of different devices.

As mentioned above, state of each LED is characterized by two bits. There are four considered LED states: off, green, red, and orange. Bit values corresponding to each of these states are written down in the Tab. 3.4.

Table 3.4: Considerd states of LED and corresponding value of state bits.

LED state	S0	S1
off	0	0
green	1	0
red	0	1
orange	1	1

Generating of Interface Files for a Client Application

To reqeust new LED states on a client side, the interface provided by *LED State Analyzer* can be used. This interface consists of one header file and one C++ source file. Both these files can be generated using *LED State Analyzer* according to the actual configuration of the test setup. Use of these files for communication with Modbus server is described in Section 4.7.1.

The content of a header file is always the same and it is copied from the file `global_data/LedStateAnalyzer_template.h`. The content of a source file depends on the configuration of the test setup. Template for it is loaded from the file `global_data/LedStateAnalyzer_template.cpp`, but line `/* INSERT DEVICES HERE */` is replaced by lines appending new instances of structure `Device` into vector `m_devices`. Member variables of the instances are defined by parameters of devices in the actual test setup configuration.

3.4.7 Documentation

For the future edits of the *LED State Analyzer*, the documentation of the entire application is included in the attachments. The documentation is in the form of a website generated by a tool *Doxygen*. All used classes and their methods are described here. The documentation can be used by open the file `documentation/index.html` in any web browser.

Chapter 4

User's Manual

This chapter contains a detailed manual for using the developed system. This chapter is mostly dedicated to a description of the user interface of *LED State Analyzer* from the user's point of view. All necessary information for correct configuration and use and also application restrictions are described in this chapter.

When using the system, the user should always proceed according to the following steps:

1. mounting of a camera holder on the rack,
2. attachment of the industrial camera to the holder,
3. adjusting aluminum arm with the camera to the suitable position (see Fig. 3.2),
4. connecting the test control system, computer with the application *LED State Analyzer* and industrial camera according to Fig. 3.11,
5. launching *LED State Analyzer*.

4.1 Requirements of *LED State Analyzer*

The user's manual is written for *LED State Analyzer* version 1.0 which was developed within the scope of this thesis. The application was tested on 64-bit Linux distribution *Mint 18*, but any 64-bit Linux distribution should be suitable to use. It is necessary to have all required libraries installed before launching or building the application (`makefile` is placed in folder `release`). List of required libraries and their versions follow:

- Gtkmm 3.0,
- pylon 5.0.11,
- libmodbus 3.0.6,
- OpenCV 3.4.0.

Installation files for all these libraries except *Gtkmm* are located in the folder `libraries` in a root directory of the application. This folder also contains file `readme.txt` describing the installation procedure or links to installation instructions provided by the authors. The application was tested only with these versions of libraries, and therefore a correct functionality with other versions cannot be guaranteed. When the application is successfully built (file `release/LED_State_Analyzer` is available and executable), a desktop link can be created by performing bash script `generate_desktop_icon.sh` located in the root directory.

4.2 Start of the Application

For communication with the test control system using a port number less than 1024, it is necessary to run the application as a root user. Bash script `generate_desktop_icon.sh` generates two desktop shortcuts: *LED State Analyzer* and *LED State Analyzer (root)*. If a user wants to use a port number requiring root access, *LED State Analyzer (root)* can be launched. Otherwise, it is sufficient to launch *LED State Analyzer*.

After a start of the application, a user is prompted to select a working directory. Selected directory is used for storing all data produced by the application including its configuration and database of devices. If the selected directory already contains application configuration and database of devices, then they are loaded. If the directory does not contain existing configuration or the files are corrupted, a default configuration from the directory `global_data` is loaded. After loading the configuration, the main window is displayed, and the chosen directory is used as a working one.

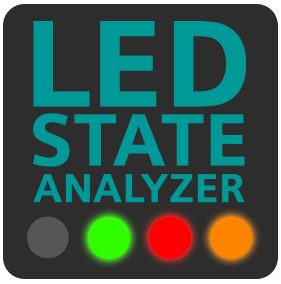


Figure 4.1: Icon of the application *LED State Analyzer*.

4.3 Preferences

Any time the application is running (except a real-time analysis), it is possible to change a configuration of image capturing and image processing. All these parameters can be changed in the window *Preferences* (Setting → Preferences). Preview of this window is shown in Fig. 4.2. Description of these parameters and their recommended values follow.

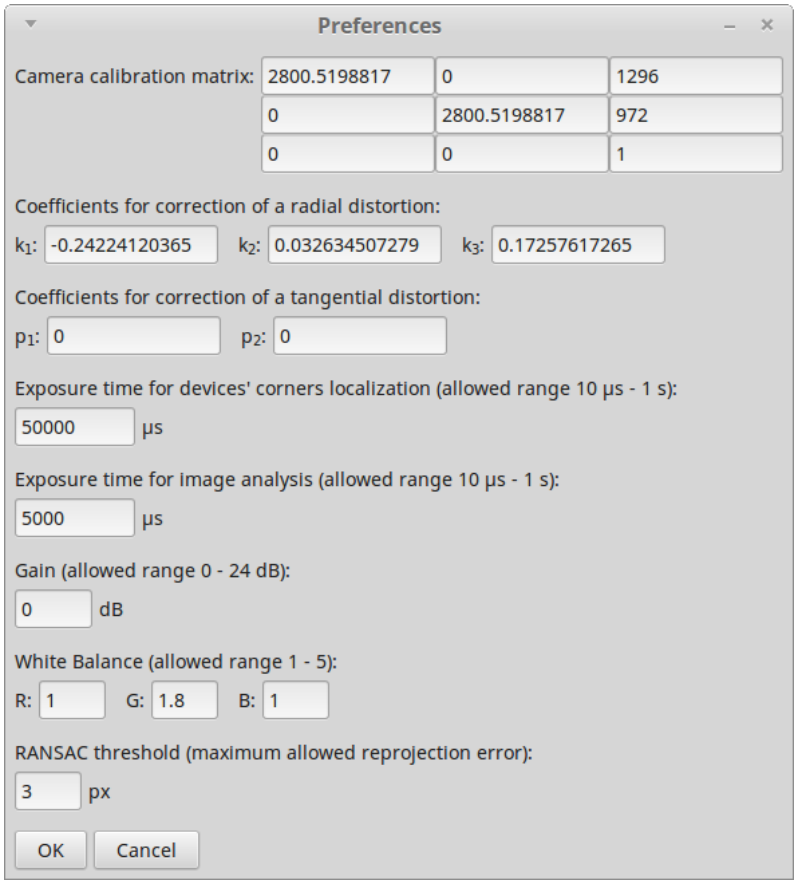


Figure 4.2: Preview of the window *Preferences*.

4.3.1 Camera Calibration

First parameters are camera calibration matrix and coefficients for correction of radial and tangential distortion. These parameters are used for suppression of camera distortion on captured images. Values filled by default were obtained by camera calibration (see Section 3.3.2). Camera calibration matrix is a 3×3 matrix which contains intrinsic parameters of the camera. This matrix and distortion coefficients are used for image adjustment to the form suitable for localization of device corners by a user and localization of LEDs by the application. It is necessary to have an image deformed as little as possible, and therefore these parameters must be precise. LEDs on devices cannot be localized sufficiently precise on a distorted image, which makes the analysis much less reliable.

4.3.2 Exposure Times Setting

Exposure forms the connection between lighting, time, and camera hardware (CCD sensor) [23]. The goal is to set exposure time optimal concerning light conditions of environment and requirements to the image. Camera *Basler daA2500-14uc* allows to set exposure time between 10 µs and 1 s [2]. Exposure time can be separately configured for two different cases when the camera captures an image.

Setting of the Exposure Time for Devices' Corners Localization

The first case is when the captured image is used for localization of device corners by a user. In this case, the set value is entirely up to the user because this value has no impact on the image analysis. It is recommended to set a value for which the user will be able to see and mark device corners easily. An exposure time of an image for feature localization can be for example somewhere between 40 000 µs and 100 000 µs. This value also depends on the opening of the iris.

Setting of the Exposure Time for Image Analysis

The second case is when the captured image is used for the analysis. A setting of this parameter is much more critical and has a significant impact on the accuracy of LEDs states recognition. Testing image, captured with the same configuration as the one for the analysis can be checked in the tab *Configuration of LED State Recognition* (see Section 4.6). Exposure time should be set to such a value that the most brightness LEDs (typically in the center of the image) should not be saturated on the image (LEDs should not look like white, but their real color should be easily recognizable). For that reason, the exposure time should be set just below saturation. However, if the time is too short, there can be a problem with recognition of LEDs which are controlled by PWM. In that case the LEDs may sometimes be captured as turned off even though they are turned on. Exposure time also depends on LEDs brightness and the set iris, and therefore the optimal value has to be found experimentally by a user. Very rough estimate and a good starting point for testing can be about 5 000 µs.

4.3.3 Gain

Gain allows the brightness of the image captured by a camera to increase. The used camera supports only setting of global gain, and therefore it is not possible to increase the brightness of only one color channel. Camera *Basler daA2500-14uc* allows to set gain from 0 to 24 dB. This camera supports only digital gain which is applied after conversion of measured values which means that these values are only multiplicated by a value according to the set gain. It means that the use of gain does not bring any additional information and it may even distort measured values. However, this feature can be useful for increasing the user's comfort during camera configuration [2].

4.3.4 White Balance

White balance allows the intensity of a color channel to increase proportionally. This feature can be used for example when colors of an image are distorted by ambient lighting. Balance ratio of each channel can be set to a value between 1 and 5. Setting a balance ratio of some channels to for example 1.2 means that the intensity of this channel is increased by 20% [2].

4.3.5 RANSAC Threshold

RANSAC method is used during object tracking in localization of devices' corners (see Section 4.5.2). Used method of object tracking and meaning of RANSAC threshold is explained in Section 3.3.4. RANSAC threshold is an acceptable error (in pixels) of mapping from one image to another. If the threshold is too small, corresponding points can be classified as outliers. Otherwise, if the threshold is too big, not corresponding points can be classified as inliers. Reasonable RANSAC threshold should be between 1 and 10 pixels.

4.4 Test Setup Configuration

The main window contains four tabs and first of them, which is opened after the start of the application, is called *Test Setup Configuration*. This tab provides an interface to configure a test setup based on the currently tested devices. Static parameters of each device are displayed and also the name of each device can be changed here. It is not allowed to switch to another tab until the industrial camera is successfully found and initialized. Preview of this tab is shown in Fig. 4.3.

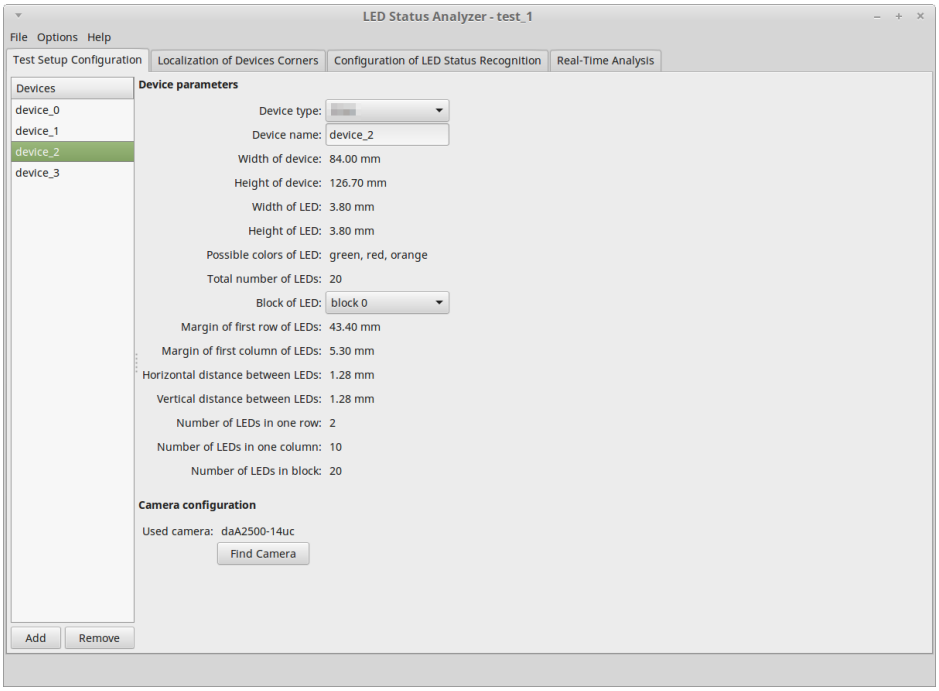


Figure 4.3: Preview of the main window with selected tab *Test Setup Configuration*.

4.4.1 List of Devices

List of devices consists of all currently tested devices. The types of devices can differ. A device can be added by the button *Add*, and the selected device can be removed by the button *Remove*. A maximal number of devices is not restricted, but the total amount of LEDs on them is. Maximal number of LEDs in one configuration is set to 800.

4.4.2 Naming of Devices

Each device has a name which can be changed by the user. A new name can be written to the text entry next to the *Device name* and has to be unique for each device.

4.4.3 Camera Connection

Before switching to another tab, it is necessary to connect an industrial camera and pressing the button *Find Camera* to establish communication with it. The application supports only one type of camera which is *Basler daA2500-14uc*. Other types of industrial cameras are not supported in this version of the application.

4.5 Localization of Devices' Corners

Tab *Localization of Devices' corners* is determined for localization of corners of all devices by the user. This step is necessary for correct LED recognition. Firstly, a new image should be captured or loaded from the file if it was taken before and the configuration of devices on the rack did not change. If the image is taken, it is saved in a working directory. Independently of the way of getting an image, the image is shown on the screen, and devices' corners localization can be performed. Preview of this tab is shown in Fig. 4.3.

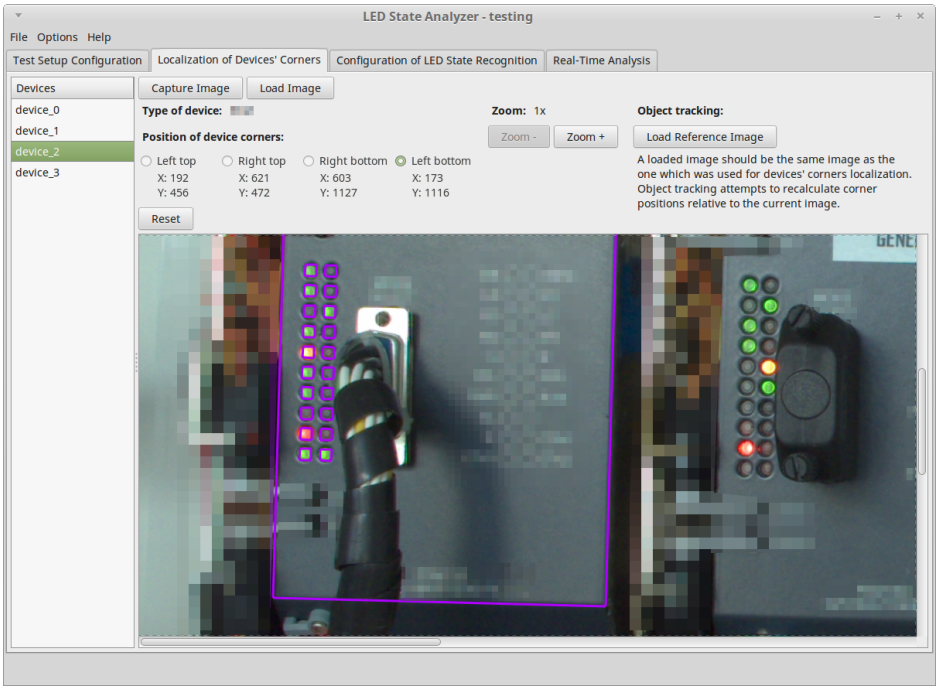


Figure 4.4: Preview of the main window with selected tab *Localization of Devices' corners*.

4.5.1 Device Corners Marking

Before configuration of LED state recognition and real-time analysis all corners of all devices shall be marked on the image. Marking corners is intuitive. Select device from the list (on the left side) → select radial button which corresponds to one of the device corners → click on the image where this corner is located. To increase the precision, zoom can be used. Image can be zoomed 1× (original image), 2×, or 4×.

When all four corners of the selected device are defined, borders of a device and frames around LEDs are shown in the image. A position of LEDs is based on static parameters of the device type. LEDs should be within frames as precise as possible, and therefore it can be necessary to change device corners location to increase the precision of frames location.

4.5.2 Object Tracking

Sometimes the devices on the rack are precisely in the same deployment used for analysis in the past but the relative position of the camera to them is different. In such situations the positions of corners from the loaded configuration do not precisely match corners positions on a new image. This is a case when using an object tracking may be appropriate. Object tracking is a process of finding homography between two images of the same object taken from different places.

To use an object tracking, a current image has to be captured and old configuration containing locations of devices' corners should be loaded. If the current image is shown, a reference image which corresponds to the loaded configuration can be loaded (button *Load Reference Image*). The process of object tracking takes a few seconds and after that the positions of devices' corners are transformed with respect to the current image. This feature can save time of devices' corners localization when the position of DUTs stays unchanged.

Note: Object tracking is computationally demanding operation and can take a few seconds (typically 1 – 5 s). Time of computation primarily depends on the performance of used computer.

4.6 Configuration of LED State Recognition

Tab *Configuration of LED State Recognition* is used to test the accuracy of LEDs states recognition and modify reference colors of each LED state. Each device can support up to three LED colors: green, red, and orange (simultaneously shining green and red). Reference color of each of these states and also a state when the LED is turned off can be configured here. Preview of this tab is shown in Fig. 4.5.

To set reference LEDs according to their real values or to test LED state recognition accuracy, a new image can be captured (button *Capture Testing Image*). A captured image is undistorted using coefficients set in *Preferences* window. A cutout of the image with the currently selected device is shown in a middle of the window.

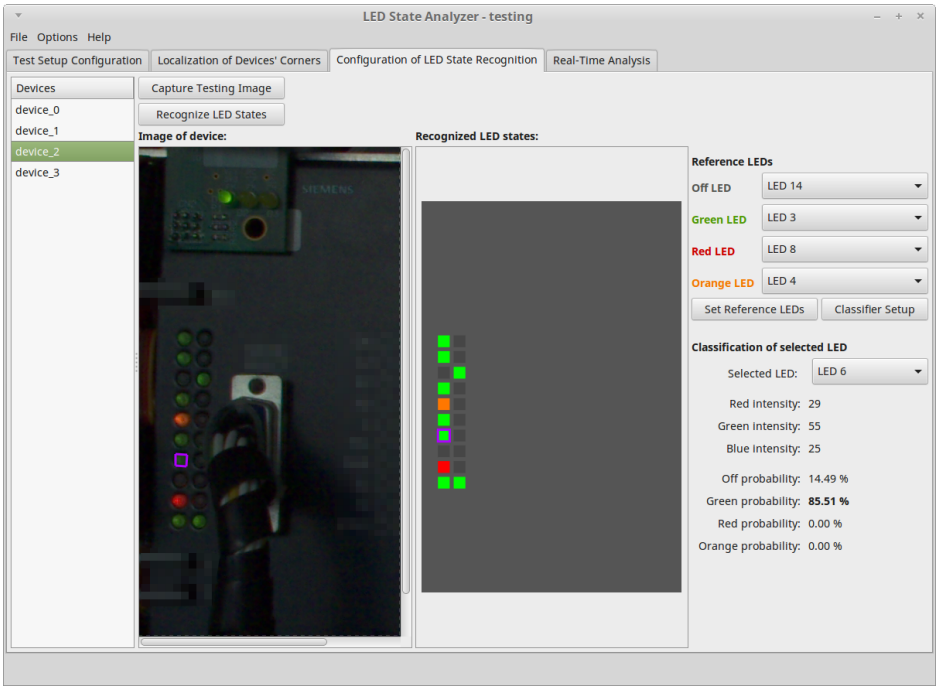


Figure 4.5: Preview of the main window with selected tab *Configuration of LED State Recognition*.

4.6.1 Classifier Configuration

The classifier is automatically configured according to the colors of reference LEDs, but it can also be configured manually. The classifier is separately configured for each device. A preferred way of configuration is to select LEDs whose average color will be used as reference values for LEDs states recognition. Selecting the reference LED for each recognized LED state can be performed in the top right part of the window (section *Reference LEDs*). After selecting the reference LEDs, their colors can be set as a reference by pressing the button *Set Reference LEDs*.

Sometimes the user may want to setup a classifier manually or at least edit its parameters. Editing the classifier parameters can be performed by pressing the button *Classifier Setup*. Preview of a displayed window is shown in Fig. 4.6.

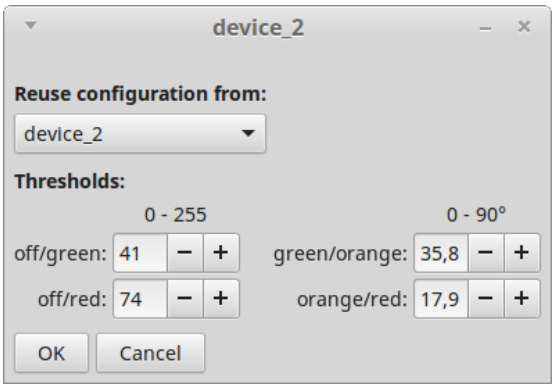


Figure 4.6: Preview of a window for classifier setup.

As the figure shows, four thresholds can be adjusted. Graphical meaning of them is shown in Fig. 4.7. It also shows how can these thresholds be changed. For correct setup a user should proceed according to the following steps:

- If a red or orange LED is misclassified as an off LED, value of *off/red* threshold should be decreased and vice versa.
- If a green or orange LED is misclassified as an off LED, value of *off/green* threshold should be decreased and vice versa.
- If a green LED is misclassified as an orange LED, value of *green/orange* threshold should be decreased and vice versa.
- If an orange LED is misclassified as a red LED, value of *orange/red* threshold should be decreased and vice versa.

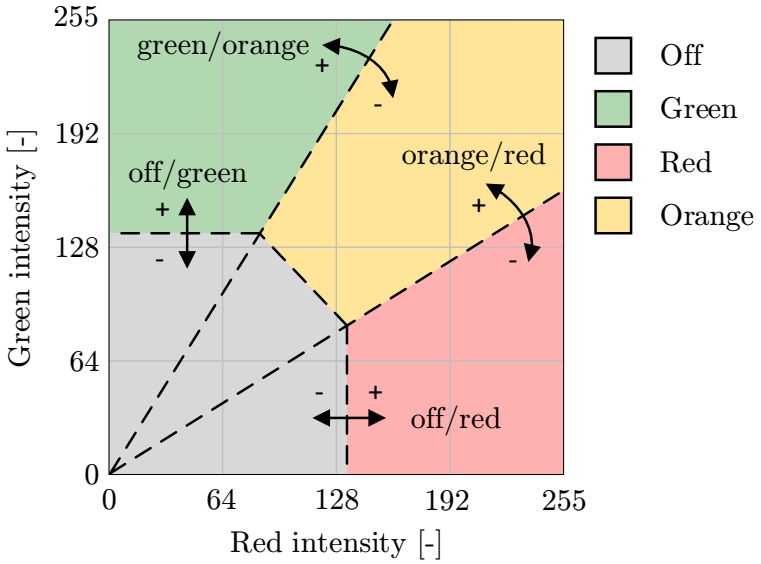


Figure 4.7: Illustration of thresholds used for classification. Thresholds *off/green* and *off/red* are defined by intensity of green or red respectively. Thresholds *green/orange* and *orange/red* are given by the angles that constrain with the x-axis (red intensity).

4.6.2 State Recognition Testing

Beside configuration of a classifier, it is also possible to test the accuracy of LEDs states recognition in this tab. Recognition of LEDs states can be performed by pressing the button *Recognize LED States* if an image is available. After recognition, the illustration of a selected device is updated, and LEDs of the device are colored based on their recognized state.

After recognition, an actual image of device can be seen together with its illustration with recognized LEDs states side by side and therefore it is easy to see if the recognition was done correctly. The whole process of LED state recognition is described

in Section 3.3, but the basic principle is that for each LED, theprobabilities of belonging to each of classes (off, green, red, orange) are calculated. LED is then classified to the class with the highest probability. The higher the difference between the highest probability and the other probabilities, the higher the robustness of the recognition. Classification process is described in Section 3.3.5.

Detailed information about classification of each LED can be seen in a bottom right part of the window (section *Classification of selected LED*). LED can be chosen using combo box and then the selected LED is marked in the image and also in the device illustration. Under the combo box, the average intensity of each color channel of selected LED and probabilities of LED belonging to particular class are shown. The highest probability is in bold.

4.7 Real-Time Analysis

Tab *Real-Time Analysis* starts a real-time analysis of LEDs states. Preview of this tab is shown in Fig. 4.8. A Modbus server has to be configured before the starting the analysis. Configurable parameters of communication are IP address and port number of the test control system. Real-time analysis can be launched by pressing the button *Start Analysis*. A process of communication between client and server is described in Section 3.4.6. After starting the real-time analysis, it is not allowed to switch to another tab or edit preferences. When the analysis is finished, a user can turn off the Modbus server by pressing the button *Stop Analysis* (available only when the analysis is running).

In the center of the window, a communication log is displayed. It shows a state of communication between the client and the server, data sent to the client, and time between receiving a request and a sent response. Data are in hexadecimal format split into 16-bit values which correspond to Modbus registers. The log can be deleted by pressing the button *Clear Log* or it can be saved to the file by pressing the button *Save Log*.

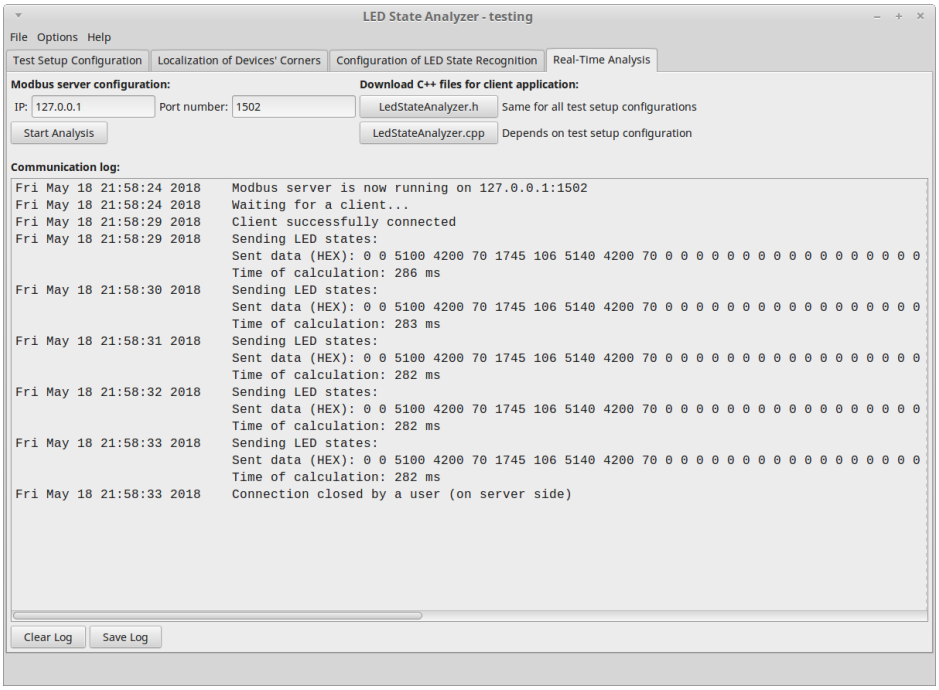


Figure 4.8: Preview of the main window with selected tab *Real-Time Analysis*.

4.7.1 Interface for Sending Requests from a Client Application

To request LEDs states from the client application, the interface provided by *LED State Analyzer* can be used. This interface consists of one C++ source file (`.cpp`) and one header file (`.h`). These files can be generated and saved by pressing the buttons *LedStateAnalyzer.cpp* and *LedStateAnalyzer.h*. File *LedStateAnalyzer.h* is the same for every configuration. File *LedStateAnalyzer.cpp* differs depending on types, names, and order of devices. A user should always generate new file *LedStateAnalyzer.cpp* after a change of test setup configuration to prevent communication problems. Generated files can be included in the client application running on the test control system. The files contain class `LedStateAnalyzer` which provides methods for update LEDs states and reading state of any LED.

Use of class `LedStateAnalyzer` is straightforward. All functionalities are provided by one instance of this class. Before creating an instance, Modbus client shall be already initialized. A constructor of class `LedStateAnalyzer` expects structure with Modbus configuration data as a parameter. After creation of the object, its member methods can be used. There are two public member methods which can be used by a user.

Method `bool updateLedState()` sends a request for data to Modbus server (application *LED State Analyzer*) and after processing the request, new data are read from the Modbus registers. These data are saved into member variable and remain unchanged until next call of this method. If a state update is performed correctly, a method returns `true`, otherwise it returns `false`.

Method `LedState getLedState(std::string name, unsigned int ledId)` returns state of selected LED. LED whose state is requested is specified by the device

name which has to be the same as the name used in the *Test Setup Configuration* in *LED State Analyzer*, and ID number of the LED. LEDs are always numbered from top to bottom and from left to right. Returned LED state is one of the values of `enum LedState`, possible values correspond to off, green, red, orange, and error. State of LED is always based on last data received from Modbus server by using method `updateLedState`. The method `getLedState` does not communicate with Modbus server and therefore returned state of LED does not have to be up to date. If it is required to get the current state, a method `updateLedState` has to be called immediately before method `getLedState`.

4.8 Generating of Database of Devices

Default database of devices types located in the folder `global_data` contains only one testing type of device. This file shall always contain at least one device type. Otherwise, an error occurs when loading it. New file `database_of_devices.json` can be generated using python script `database_of_devices_generator.py` located in the root application directory. Some parameters of device type are described in this script, rest of them are described in Appendix A. `database_of_devices.json` located in the folder `global_data` is used when a new project is created. Every saved project contains its database which can differ among the projects.

Chapter 5

System Testing

The system was tested under real conditions. Testing procedure was done according to the user's manual in Chapter 4. DUTs used for the tests were the same type as devices used during development. Devices of a different type were not available and therefore they could not be used. However position of devices under test, their number and also light conditions was different. Test setup used during testing is shown in Fig. 5.1.

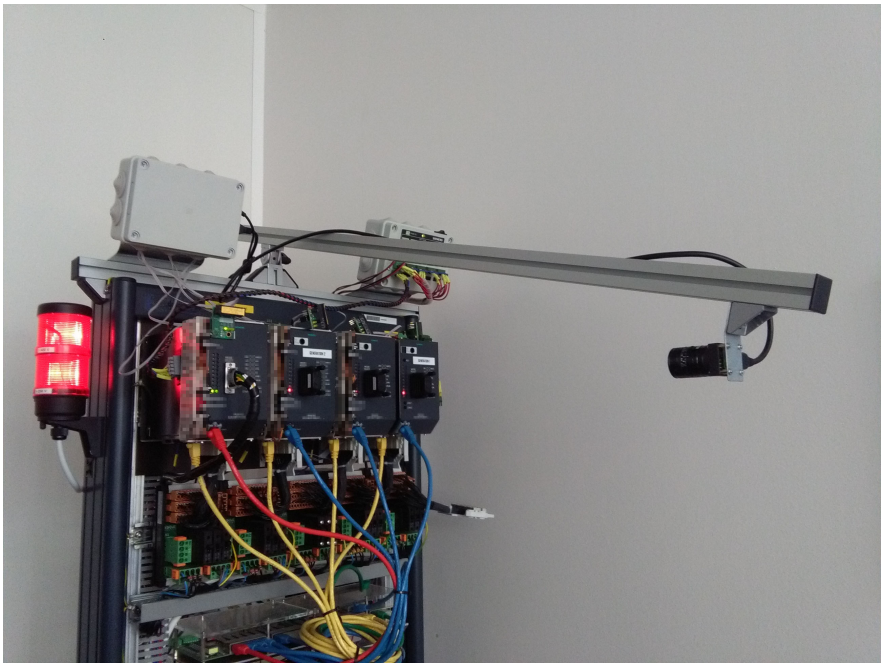


Figure 5.1: Test setup used during testing with attached industrial camera.

5.1 Mechanical Arrangement and Wiring

First of all, it was necessary to mount mechanical construction to the rack and connect the computer performing the analysis with the industrial camera and test control system. Assembling of a mechanical construction and its mounting on a rack with DUTs is straightforward and it was done according to the description in Section 3.1.2. Construction is sufficiently rigid and adjustable arm makes it possible to move the camera

away when a test is not running.

Wiring of the system was done according to the diagram in Section 3.1.5. A connection between an industrial camera and the computer performing the analysis (USB 3.0) provides sufficient performance for transmission of images every 500 ms. A communication channel between the computer performing the analysis and the test control system is also sufficient for transmission of information about LED states.

5.2 Configuration of *LED State Analyzer*

A configuration of the application *LED State Analyzer* based on actual test setup was performed according to the user's manual in Chapter 4.

After initialization of a camera and adding DUTs to the list of devices, a user is prompted to localize corners of these devices on the image. If a camera calibration is done correctly, it is easy to set a sufficiently accurate frame around devices so that all LEDs are localized correctly. When a previously saved configuration of a test setup is used but a camera was moved since the time of its creation, object tracking instead of a repetition of manual localization of device corners on the image can be used. The object tracking could not be used in the real-time analysis, because it is very time-consuming, and also its accuracy is uncertain. The measured time duration of an object tracking between two 5 MP images was 1.58 ± 0.03 s.

Next step is a configuration of LED state classification. The classifier is trained based on reference color of each LED state extracted from the image. If it is not possible to use reference LEDs for classifier training, it can be configured manually. A possibility of manual configuration was the main reason for selecting a classifier composed of multiple linear classifiers and, therefore, it is easily configurable by the user.

5.3 Response Analysis of the System

Real-time analysis of LEDs is the most time-critical part of the system. LED state recognition has to be fast and accurate. Time performance of a real-time analysis was evaluated by measuring a time duration of critical processes during analysis. Time durations shown in Tab. 5.1 were calculated from one hundred measurements. Measured times are strongly dependent on the performance of a used computer, and therefore it will differ based on hardware. Tests were performed on laptop with CPU Intel Core i7-4810MQ and 16 GB RAM.

Table 5.1: Time durations of critical parts of a real-time analysis.

Process	Time duration [ms]
Camera configuration	23 ± 1
Image capturing	217 ± 4
Image pre-processing	33 ± 2
Analysis of LEDs states (200 LED)	9 ± 2
Total response (measured on a client side)	283 ± 5

The table shows measured time duration of the most critical processes of real-time analysis and also a total response time measured in a client application. Time of camera configuration is mostly given by a camera firmware and camera API, and it cannot be significantly decreased. Majority of the overall time duration takes an image capturing. This time is given by limits of USB 3.0 and a camera firmware, and there is no possibility of decreasing it. The time duration of an image pre-processing includes saving of captured image and its conversion to appropriate format for subsequent analysis. The actual analysis of LEDs states takes the least amount of time. It is optimized for the best performance and all calculations which do not have to be done during the real-time analysis are performed during the configuration process. Measured time corresponds to the analysis of 10 devices, each of them has 20 LEDs, where each LED is shown on about 217 pixels. Time of analysis of LEDs states depends on the total number of LEDs and their sizes.

The duration of a total response measured in the client application (time between sending a request for current LED states and receiving a response) is about 283 ms which meet the system requirements that this time has to be less than 500 ms.

5.4 System Stability and Accuracy

Communication between the computer performing the analysis and the test control system is stable and reliable. On the client side (test control system), there should always be an interface provided by *LED State Analyzer* used. This interface provides methods for easy update and reading of LED states.

One of the most critical properties of the system is the accuracy of LED state recognition. It was measured using a stress test. First of all, the application was configured. The test was performed under other lighting conditions than during a configuration. Before the test was launched, the camera was shifted, and positions of the devices' corners were automatically corrected by using object tracking feature. The test was launched after performing these actions. It lasted about one hour, and accuracy of LED recognition was 100 %.

Chapter 6

Conclusion

The goal of this thesis was to design and implement a system for automatic recognition of LEDs states of devices under test. States of diagnostic LEDs are recognized from the images captured by camera *Basler daA2500-14uc*. All interactions with the user are performed by the *LED State Analyzer* application. This application provides interface for configuration of the camera, LED recognition, and also communication with the test control system. Communication with the test control system is via Modbus protocol.

The developed system is fully functional and meets all requirements defined by our customer. The system was tested under real conditions, and it was found sufficiently reliable and accurate. One part of the thesis is dedicated to a user's manual which contains all information necessary to configure and use the system.

Restrictions

Although the system meets all requirements, it also has some restrictions. These restrictions are mostly caused by adaptation of the system for using it with the devices defined by our customer. The main restriction is that the only supported colors of LEDs are green, red, and orange. The classifier is adapted for recognition of these colors and, therefore, it cannot be easily changed for recognition of other LED colors. There is also restriction related to the shape of DUTs. Only rectangular shape of a device is supported. A rectangle is sufficient for most of the applications, but it can be limiting for some unusual shapes of devices.

Using an object tracking was considered in real-time analysis during development. It was found that it is not possible because of its time duration, which is much longer than the maximal allowed response time. Therefore, an object tracking can be used only before the start of a real-time analysis.

Possible Improvements

The system and especially the *LED State Analyzer* application could be improved in many ways. The application could be extended to support more general devices with

different types of LEDs with more colors. Object tracking could also be improved for use in real-time analyses. For example, images with lower resolution could be used which should increase the speed of object tracking but it could also decrease its accuracy.

The client-side interface provides the greatest space for improvement. At this point, it is possible to obtain only current state of each LED on each DUT. The possibilities of this interface could be extended for example to detect LED flashing, to determine flashing frequency and a duty cycle. However, these improvements can be made in the client application according to current requirements and without any interventions to the application on the server-side.

Bibliography

[1] *Photo of Basler dart daA2500-14uc* [online]. [Accessed 6 December 2017]. Available from: https://cz.mouser.com/images/basler/lrg/CS_Mount_SPL.jpg

[2] Basler dart USB 3.0 – Product Documentation. *Basler AG* [online]. v09. [Accessed 15 January 2018]. Available from: https://www.baslerweb.com/en/sales-support/downloads/document-downloads/basler-dart-usb-3-0-users-manual/

[3] How does the YUV color coding work?. *Basler AG* [online]. [Accessed 6 December 2017]. Available from: https://www.baslerweb.com/en/sales-support/knowledge-base/frequently-asked-questions/how-does-the-yuv-color-coding-work/15182/

[4] Basler Lens C125-0618-5M F1.8 f6mm - Lenses. *Basler AG* [online]. [Accessed 16 December 2017]. Available from: https://www.baslerweb.com/en/products/vision-components/lenses/basler-lens-c125-0618-5m-f1-8-f6mm/

[5] Documentation Overview. *gtkmm - C++ Interfaces for GTK+ and GNOME* [online], [Accessed 5 December 2017]. Available from: https://www.gtkmm.org/en/documentation.html

[6] pylon Linux x86. *Basler AG* [online]. [Accessed 2 December 2017]. Available from: https://www.baslerweb.com/en/products/software/pylon-linux-x86/

[7] Modbus Specifications and Implementation Guides. *Modbus Organization* [online], [Accessed 11 December 2017]. Available from: http://www.modbus.org/specs.php

[8] RAIMBAULT, Stéphane. A Modbus library for Linux, Mac OS X, FreeBSD, QNX and Win32. *libmodbus* [online]. [Accessed 2 January 2018]. Available from: http://libmodbus.org/

[9] Introducing JSON. *JSON* [online]. [Accessed 2 February 2018]. Available from: https://www.json.org/

[10] LOHMANN, Niels. JSON for Modern C++. *GitHub* [online]. [Accessed 4 April 2018]. Available from: https://github.com/nlohmann/json

[11] OpenCV library. *OpenCV* [online]. [Accessed 2 April 2018]. Available from: https://opencv.org/

[12] JAYARAMAN, Subramania, ESAKKIRAJAN, S. and VEERAKUMAR, T. *Digital image processing*. New Delhi: Tata McGraw Hill Education, 2009. ISBN 978-0-07-014479-8.

[13] JÄHNE, Bernd. *Digital image processing*. 6th rev. and ext. ed. New York: Springer, 2005. ISBN 3-540-24035-7.

[14] SONKA, Milan, HLAVAC, Vaclav and BOYLE, Roger. *Image processing, analysis, and machine vision*. 3rd ed. Toronto: Thompson Learning, 2008. ISBN 0-495-08252-x.

[15] KAEHLER, Adrian and BRADSKI, Gary R. *Learning OpenCV 3: computer vision in C++ with the OpenCV library*. Sebastopol, CA: O'Reilly Media, 2017. ISBN 978-1-491-93799-0.

[16] HARTLEY, Richard and ZISSERMAN, Andrew. *Multiple view geometry in computer vision*. 2nd ed. New York: Cambridge University Press, 2003. ISBN 978-0-521-54051-3.

[17] Camera Calibration and 3D Reconstruction. *OpenCV* [online]. [Accessed 8 February 2018]. Available from: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[18] AKAZE and ORB planar tracking. *OpenCV* [online]. 18 December 2015. [Accessed 7 February 2018]. Available from: https://docs.opencv.org/3.1.0/dc/d16/tutorial_akaze_tracking.html

[19] KARAMI, Ebrahim, PRASAD, Siva and SHEHATA, Mohamed. *Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images*. St. John's, Canada: Memorial University, 2015.

[20] ALCANTARILLA, Pablo F., NUEVO, Jesús and BARTOLI, Adrien. *Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces*. Atlanta, GA, USA: Georgia Institute of Technology, 2013.

[21] ANDERSSON, Oskar and MARQUEZ, Steffany R. 2016. A comparison of object detection algorithms using unmanipulated testing images: degree project. Stockholm, Sweden: KTH Royal Institute of Technology. pp 31.

[22] LOWE, David G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*. 2004, 60(2), 91-110.

[23] HORNBERG, Alexander. *Handbook of machine and computer vision: the guide for developers and users*. Weinheim: Wiley-VCH, 2017. ISBN 978-3-527-41339-3.

Appendix A

Dimensions Describing a Device

All dimensions used for a description of devices in *LED State Analyzer* are shown in Fig. A.1. Similar names of dimensions are also used in file `database_of_devices.json`. The figure shows only one block of LEDs, but one type of device can contain many rectangular blocks of LEDs. One device type supports only one type of LEDs.

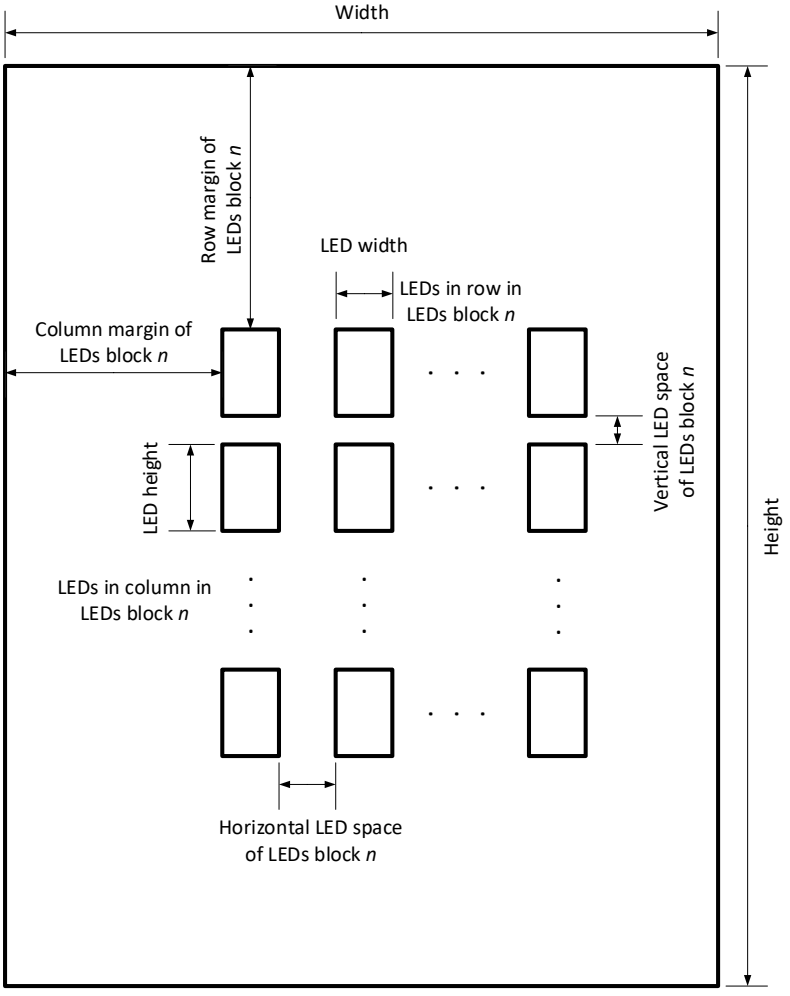


Figure A.1: General device with marked dimensions.

Appendix B

Attached CD Contents