BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING

FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ÚSTAV AUTOMATIZACE A INFORMATIKY

VISUAL INSPECTION OF PCB INSTALLATION

VIZUÁLNÍ KONTROLA OSAZENÍ DPS

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR AUTOR PRÁCE	Bc. Egor Cherniaev
SUPERVISOR VEDOUCÍ PRÁCE	prof. Ing. Radomil Matoušek, Ph.D.

BRNO 2024

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Egor Cherniaev**
Studijní program:	Aplikovaná informatika a řízení
Studijní obor:	bez specializace
Vedoucí práce:	**prof. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2023/24

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Vizuální kontrola osazení DPS

Stručná charakteristika problematiky úkolu:

Kontrola kvality osazení DPS (desky plošných spojů) je klíčovou kontrolou jakosti jejich výroby. Tuto činnost lze kromě lidské inspekce provádět ekvivalentně s využitím počítačového vidění. Strojová kontrola může být v mnoha ohledech efektivnější jak z hlediska metodologie měření, tak z hlediska nákladů. Podstatou této práce bude rešerše a vlastní návrh systému pro kontrolu zvolených parametrů kontroly osazených el. součástek.

Cíle diplomové práce:

• Rešerše průmyslových systémů strojového vidění.
• Popis zvoleného systému CV.
• Návrh řešení pro definovanou kontrolu osazených el. součástek ve vymezené oblasti DPS.
• Vlastní řešení, popis a implementace řešení.
• Experimenty a závěr.

Seznam doporučené literatury:

SZELISKI, Richard, 2010. Computer Vision: Algorithms and Applications. London: Springer. Texts in computer science. ISBN 978-1-84882-934-3.

BATCHELOR, Bruce G., ed., c2012. Machine vision handbook. London: Springer. Springer reference. ISBN 978-1-84996-168-4.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2023/24

V Brně, dne

L. S.

doc. Ing. Zdeněk Hadaš, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRACT

Visual inspection of PCBs is a significant area of development in applied informatics in the electronics industry. The presented work is dedicated to a theoretical exploration of algorithms used for detecting specific assembly defects of components on printed circuit boards and the practical implementation of a system for automating general visual inspection. The exploration primarily focuses on cases of missing components, capacitor orientation, and print quality on the board. The practical part proposes an own inspection system based on the ROS2 framework. The paper describes various aspects of the development and utilization of the system. An integral part of the work is a publicly accessible dataset comprising over 300 images of PCBs captured using an industrial camera. The conclusion includes an assessment of the contribution of the presented work.

ABSTRAKT

Vizuální kontrola DPS je významnou oblastí rozvoje aplikované informatiky v elektronickém průmyslu. Předložená práce je věnována teoretickému průzkumu používaných algoritmů pro identifikaci defektů montáže komponent na desky plošných spojů a praktické implementaci systému pro automatizaci obecné vizuální kontroly. Průzkum je primárně zaměřen na případy chybějících komponent, orientaci kapacitorů a kvalitu potisku na desce. Praktická část je věnována návrhu a implementaci vlastního systému na frameworku ROS2. Práce popisuje různé aspekty vývoje a použití daného systému. Součástí práce je veřejně přístupný dataset s více než 300 snímky DPS získanými průmyslovou kamerou. Závěrem je provedeno hodnocení přínosu předložené práce.

KEYWORDS

machine vision, automated optical inspection, pcb, automation, ros2

KLÍČOVÁ SLOVA

strojové vidění, vizuální kontrola, dsp, automatizace, ros2

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

2024

BIBLIOGRAPHIC CITATION

CHERNIAEV, Egor. *Visual inspection of PCB installation.* Brno, 2024. Available at: https://www.vut.cz/studenti/zav-prace/detail/161094. Master's Thesis. Brno University of Technology, Fakulta strojního inženýrství, Ústav automatizace a informatiky, Supervised by prof. Ing. Radomil Matoušek, Ph.D.

AUTHOR'S DECLARATION

I declare that I have independently prepared my diploma thesis on the topic 'Visual inspection of PCB installation' using the cited literature and sources.

V Brně dne 22. 5. 2024 ....

Egor Cherniaev

ACKNOWLEDGEMENT

I would like to express my gratitude to Prof. Ing. Radomil Matoušek, Ph.D. for his valuable advice, recommendations, and patience in guiding my diploma thesis. I would also like to thank Prof. Ing. Pavel Škrabánek, Ph.D. for his friendliness, willingness, and assistance in obtaining the necessary information and materials.

CONTENTS

1 Introduction

Printed Circuit Boards (PCBs) play a critical role in today's world. They are found everywhere, from mass-produced consumer products to highly unique and expensive machines and mechanisms. More often than not, PCBs are critical components within systems. An unexpected failure of such a component can lead to losses for all related parties. For customers, the failure has the most direct consequences in the form of personal injuries, direct and indirect costs (reputation, loss of market, payoffs to clients, etc.) [1]. For the manufacturer, the failure is related to warranty costs, customer losses, claims for damages, etc. [1]. These factors drive improvements in the electronics industry's quality control processes.

The quality control is generally understood as a system of precautions and direct actions to prevent nonconformity (defective product). In modern understanding, quality requires the participation of all parties related to the final product, from vendors through managers on the site to end consumers [2]. The quality control philosophy tends to ensure quality through improving production process stability. Compared to earlier attempts to improve quality through total testing, this approach is more dedicated to detecting the root cause of nonconformity rather than dealing with consequences in the form of defects. However, this does not mean that inspection and testing should be abandoned. Inspection and testing have an important role in catching occasional defects before the product is out on the market, as well as providing valuable insights into process stability. The need to catch defects as early as possible is related to the cost of the defect and the amount of rework needed to fix the defect.

The PCB manufacturing is known to be a complex process consisting of two main parts: bare board production, where etching defects are common, and board assembly, where solder joint defects and component-related defects dominate. To catch defects as early as possible, the inspection process is designed as a cascade of inspection stages that follow major production steps, such as etching or component placement. The traditional approach for inspection is manual control with human inspectors. However, this has been proved to be inefficient [3]. Instead, automated inspection systems have been developed.

The optical inspection is considered one of the most challenging tasks to accomplish. This explains the dominance of human inspectors in this area for a long period of time. The evolution of inspection systems is related to the increasing computational power and achievements in applied informatics. There have been many attempts to apply machine vision for PCB inspection. The most common approach is to compare the tested board with a so-called golden board or template. Modern approaches seem to rely more on machine learning and deep learning. However, the general features of inspection systems remain the same; each inspection system contains at least one camera, computer, and actuator. Many variations of inspection systems are available today, but generally, inspection is conducted online, meaning that the board is inspected right after it is produced.

While there are many commercial systems on the market, they may not always fit the needs of the manufacturer. Potential downsides of such commercial systems include high prices, vendor dependency, lack of options for system modifications, licensing restrictions, etc. Given this, a manufacturer may opt for a custom inspection solution. However, custom-coded solutions are not perfect as they require a lot of work to create and maintain. Considering the fact that, for most manufacturers, the basic demanded features of such systems are generally the same, developing a more general yet customizable solution becomes more logical.

This work is dedicated to the development of an optical inspection system, which will cover the basic needs of many manufacturers (not only PCB). The work describes the theoretical basis of optical inspection and then proceeds with the implementation of such a system. The proposed system is developed as an open-source project under the name Open AOI. The system is designed to be used with customer cameras and to run on customer hardware. It provides a GUI to manage collected data and set up inspection flows. The system is designed to be customizable and modular, allowing for the running of custom code and distribution across a network if required. Open AOI uses no commercial code and may become the ideal solution for low-volume PCB production.

See the difference with Open AOI!
Cherniaev E.

2 Printed Circuit Boards

Printed circuit boards serve as the foundation in electronic packaging, constituting the interconnection medium upon which electronic components are integrated into electronic systems [1]. The IPC (Institute of Printed Circuits) is a trade association dedicated to standardizing the assembly and production requirements of electronic equipment and assemblies. In the IPC-T-50N standard, IPC defines a PCB as a printed board that facilitates both point-to-point connections and printed components arranged in a predetermined configuration on a common base [4].

PCBs are classified based on the following criteria [1]:

- **Dielectric materials**: epoxy, bismaleimide triazine, and others.
- **Reinforcement materials**: glass fabric, kevlar fabric, and others.
- **Circuit type**: digital, analog, mixed, RF, and others.
- **Component mounting technology**: through-hole, surface mount, mixed.
- **Board construction**: single-sided, double-sided, multilayer, flex, rigid-flex.
- **Design complexity**

2.1 Production Process

The general production process is schematically described by Figure 1. The inner layer process begins with the selection of a core (laminate). The core is made up of fully cured epoxy sandwiched between two layers of copper cladding. The copper cladding will eventually form two inner layers, and the laminate will act as the dielectric spacing between these layers. Etch resistant is then placed on both sides. The desk is then exposed to UV light through the artwork, which creates a circuit image on both sides. Non-polymerized film is then removed. The next step is etching. The core is passed through an ammonical etch and exposed, and unwanted copper is removed. The etch resistant is then removed. Automated optical inspection is performed at this stage. [1]

Once all required cores are ready, they are stacked to form a "blank". Copper foil is placed on top and bottom. Prepreg is used to glue layers and to form the necessary dielectric between adjacent cores. [1]

Outer layer process starts with drilling. X-ray and visual inspection are used to verify drill quality and hole alignment. Deburring equipment removes any burrs that may have formed. A thin copper layer is then formed on the board surface with an electroless plating process (including hole surfaces). Circuit pattern is then formed using plating resistant. The resistant is cured under UV light using artwork that will allow curing where no copper is wanted. More copper is then deposited on the traces, pads, and hole barrels electrolytically. Plating resistant is then removed.

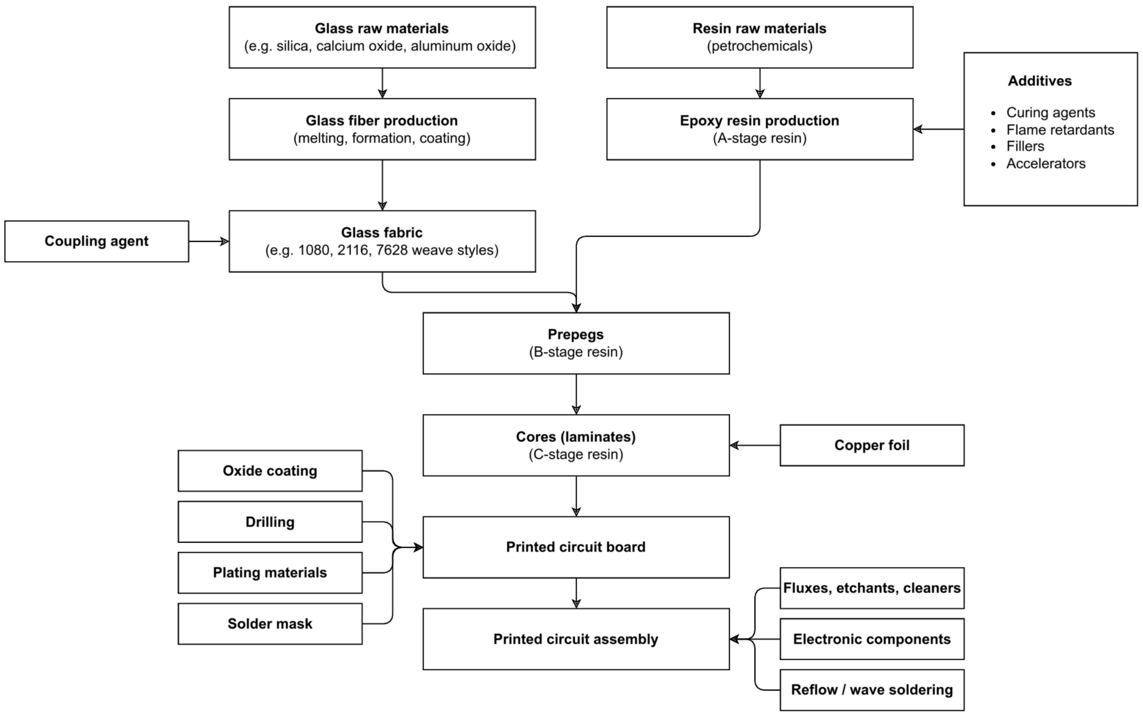


Fig. 1: General production process of PCB [1].

The copper foil is now etched away, using an ammoniacal solution, and the PCB circuit is born. After that, solder mask is applied to protect and insulate the circuitry. Hot Air Solder Level (HASL) involves the application of solder to selected board features wherever copper was left exposed after solder masking. Information needed by the customer for assembly or troubleshooting is screen-printed onto the board. The nickel/gold plating process is one of the final stages in the bare board production cycle. [1]

PCB component assembly is the process of soldering or assembling electronic components to a printed circuit board. A PCB assembly process may use through-hole assembly technology or surface mount technology (SMT), or a mix of both. Using through-hole technology, the component leads are inserted into the holes drilled in the PCB. In the SMT method, the electrical components are mounted directly onto the surface of a printed circuit board (PCB). In industry, this approach has largely replaced the through-hole technology construction method of fitting components, largely because SMT allows for increased manufacturing automation, which reduces costs and improves quality.

2.2 PCB Defects

Various defects may occur at the manufacturing stage, which may severely affect the functionality of the PCB. For example, during the component placing operation, common defects are missing or wrong components, misaligned components, and others. During soldering (reflow stage), solder joint-related defects may occur, such as pseudo joints, excess solder, insufficient solder, and others. Table 1 summarize most frequent defect types.

PCB Part	**Defect Type**
Solder joint	Missing solder, insufficient or open solder, excess or bridged solder, pseudo solder (cold solder), solder joint overall quality, solder paste inspection
Electronic Components	Missing component, wrong component, shifted or rotated component, component (tombstone) lifted, component misplacement, IC molding surface, electronic component overall quality
PCB Holes	Microdrill bits
Other	Golden fingers, traces and inner layers, marking inspection, cosmetic and small defects, scratches, improper etching, glue quality, flux cutting, alignment and position inspection of PCB, PCB overall quality, ball (BGA) grid array

Tab. 1: PCB defect types.

The distribution of defects is reported to be generally as follows, according to [3]: 71% of defects are related to soldering, wrong components, component misplacement, and other component defects are accountable for 15% in total, bridges have 7%, and other defects also account for 7%.

2.3 Quality Inspection and Testing

The widely used standard for PCB quality control is the IPC-A-610 standard, established by the Association of Connecting Electronics Industries (a standard aimed at ensuring electronic assemblies quality). This standard offers comprehensive acceptance criteria for various aspects of PCB assembly, including soldering and component placement. Another important standard is ISO 9001, which specifies the requirements for a quality management system. This standard focuses on the processes and procedures employed in PCB manufacturing.

2.3.1 Product Inspection Approaches

Optical inspection (visual-based approaches)
Detects surface flaws and defects, performed by a human inspector (manual inspection) or image processing system (automated inspection). It is indeed the most basic, relatively low-cost, and frequently used among all quality control approaches. Considered non-contact and non-destructive, this approach is not efficient for detecting inner defects and for inspecting densely populated boards. It can be deployed

inline at several points in the manufacturing process (bare board inspection, final assembly inspection, etc.). [3] [5]

X-ray inspection (AXI)
Inspection system typically consists of three main parts: X-ray source, detector, and a fixture to hold and control the position of the inspected component. The approach is suitable for surface and inner inspections (voids and crack defects in solder joints, for example). However, the method may be destructive, and inspection resolution is low (at the macro level). [3] [5]

2.3.2 Product Testing Approaches

After a PCB has been manufactured and has passed all inspections, it is ready for testing. In-circuit testing (ICT) and functional testing (FCT) are the most common test methods [5].

In-circuit testing
In-circuit testers locate faults by checking components and circuit paths locally, one-by-one [6]. Populated PCBs are checked for shorts, opens, resistance, capacitance, and other factors to assert correct assembly (in the 1980s and 1990s, workmanship problems were estimated to be responsible for 85-90% of board faults [6]). ICT can be performed with so-called "bed-of-nails" test fixtures (a device equipped with an array of small, spring-loaded pogo pins) or with a flying probe setup (where a moving probe makes contact with the board where required). The bed-of-nails approach is more expensive, and changes are difficult to make. This test does not power the tested unit (avoiding possible damage) and does not perform overall circuit functionality verification. [5]

Functional testing
Functional testing measures the performance of the board as a whole. Functional testers are highly sophisticated. The test intends to replace "hot-bed" testing by placing the board in close to an operational environment. The overall tester system usually consists of a cabinet, an interface to the DUT, cabling to connect all of the instruments, a central processing unit, and monitors. The test identifies functional defects. Functional testing is expensive and requires a deep understanding of the tested unit and its operational environment. Expensive high-speed equipment is also required. [6] [5]

2.4 Automated Optical Inspection

Quality control procedures may generally be classified as manual, automated, or semi-automated depending on production requirements. Optical inspection is one of the most common inspection types in the industry (along with Automatic Laser Measurements, X-Ray Inspection, In-Circuit Test, and others). It is one of the most difficult inspection tasks. Optical inspection seeks to identify both cosmetic and functional defects. [3] [7]

Manual visual inspection is historically the earliest type of optical inspection. The process usually involves a human inspector, whose job is to identify a wide range of possible defects on every board. Semi-automated optical inspection offers a slightly improved approach by utilizing a camera system and special software environment. However, manual inspection is generally inefficient, less reliable, and thus more expensive compared to the automated process. Human vision inspection capabilities decline with dull and routine jobs, which affects the accuracy of inspection. Multilayer board inspection is also not feasible with this approach, as well as tolerance control. On the other hand, the automated approach removes subjective aspects from inspection, quantifies results, increases inspection rates, and reduces costs. The development and integration of image capturing and processing techniques in recent years have caused a focus shift towards automated optical inspection. This trend is seen to be proceeding. [7] [3] [8]

On the SMT production lines the AOI stations are installed after component mounting and after reflow operations. Sometimes the AOI machine is only placed after the reflow oven, which is less expensive, although defects after reflow oven costs the manufacturer much more rework. [9]

3 Quality and Reliability

One of the main factors that drives a product's commercial success is its quality [10]. This section introduces general quality-related terminology and theory. It also explains the quality position in the manufacturing industry.

3.1 Quality Philosophy

Quality has been defined in many different ways. A frequently used definition of quality was once provided by Crosby, P.B. (1979), as conformance to requirements or specifications. A more general definition was earlier proposed by Juran, J.M. (1974), who defined quality as fitness for use. The intended level of quality is measured using **quality characteristics** (structural, sensory, time-oriented, ethical, etc.), which come in the form of **variables** (measurable characteristics, may be expressed on a numeric scale) and **attributes** (characteristics that are non-measurable, being either confirming or non-conforming to specification). In terms of quality, a **defect** is a characteristic that does not meet certain standards (the modern term for defect is **nonconformity**). [2]

As the quality of a product or service is a crucial factor for market success, quality improvement becomes an important part of the production process. Whereas process control deals with the identification and elimination of **special causes** (identifiable reasons can be determined), quality improvement is related to the detection and elimination of **common causes** (which have a uniform impact on the result and are always present in a system). Quality improvement is seen today as a result of a joint effort between an operator (mostly responsible for special causes) and management (mostly responsible for common causes). This process is not limited to meeting specification requirements but also includes the reduction of system variability and reducing deviation from standards. [2]

Quality is tightly related to manufacturer expenses over the product lifetime. The termination of a device or system's ability to perform its function is known as **failure**. Brindley K. [6] states that reliability improvement may lead to expense reduction and may allow the product to achieve higher quality standards. It is also stated that preventing defects at early production stages incurs less overall rework amount, which greatly reduces manufacturing costs (roughly by a factor of ten and more for complex assemblies). Failures during product consumption (external failures, [2]) are known to be especially expensive. For example, in the years 2002-2005, the National Aeronautics and Space Administration (NASA) reported over $10B in losses due to failures in field [1]. Failures do not only happen to hardware; another example is related to JetBlue Airlines. The company experienced service downtime due to the fact that on February 14, 2007, the New York JFK airport was covered with two inches of ice. The incident was fully resolved only a week later, causing the cancellation of nearly a thousand flights and reputation losses [10].

3.2 Reliability of a System

Quality is often confused with reliability, and these terms are used interchangeably, even though one does not necessarily imply the other. While quality is the ability of a system to conform to specified performance, **reliability** is a system's ability to perform its required function under known conditions for a certain period of time. [6] [1]

With system reliability being a subject for measurement, a few important metrics could be defined and used for potential maintenance purposes [6]. **Failure rate**, λ, is the frequency with which a system or component fails, expressed as the number of failures over a time unit. The average time between failures is known as **mean time to failure (MTTF)**:

$$MTTF = T/N = 1/\lambda, \tag{1}$$

for multi-component system:

$$MTTF = nT/N, \tag{2}$$

where T is the total operating time in hours, N is the number of failures, and n is the number of components. The time it takes to isolate and repair a failed system or component is known as **mean time to repair (MTTR)**, and it is an important measure of the system's **maintainability**. Together, MTTF and MTTR describe how often a failure can be expected, which is known as **mean time between failures (MTBF)**:

$$MTBF = MTTF + MTTR \tag{3}$$

Generally, three periods may be observed during a system's lifetime (this is often illustrated as the bathtub curve). During the **early period**, the failure rate is high, and failures are mainly caused by substandard components or workmanship. Such failures are called infant mortality [1]. During the **useful life period**, the failure rate is constant, and the system's MTBF is related to failures in this period. A final increase in the failure rate is often observed during the **wear-out period** and is caused by system deterioration with age. [6]

3.3 Reliability of a PCB

In this section, PCB defects and failure mechanisms are discussed. The main goal is to trace a PCB failure back to its main root cause and identify the role of PCB testing processes during manufacturing in failure prevention.

As stated earlier, failure is the inability of a system to perform its required function. The physical, chemical, thermodynamic, or other processes that result in failure are known as **failure mechanisms**. In exploring PCB reliability, NASA [1] defined two main classes of failure mechanisms.

3.3.1 Overstress mechanisms

Overstress failure is expected when use conditions exceed the system's dimensions defined during design (like material strength limit). Failure is often sudden and catastrophic. Common mechanisms include:

- **Mechanical**: yield, fracture, interfacial de-adhesion.
- **Thermal**: glass transition, phase transition.
- **Electrical**: dielectric breakdown, electrical overstress, electrostatic discharge, second breakdown (bipolar transistors).
- **Radiation**: single event upset (SEU).
- **Chemical**: contamination.

3.3.2 Wearout mechanisms

Wearout failure is typical where damage accumulates over time or with repeated stress. Common mechanisms include:

- **Mechanical**: fatigue, creep, wear.
- **Thermal**: stress-driven diffusion voiding (SDDV, also known as stress-induced voiding, SIV, or stress migration).
- **Electrical**: TDDB, electromigration, surface charge spreading, hot electrons, CAF, slow trapping.
- **Radiation**: radiation embrittlement, charge trapping in oxides.
- **Chemical**: corrosion, dendrite growth, depolymerization, intermetallic growth.

For some mechanisms, the root cause of failure may be traced back to the manufacturer. This is the case for chemical contamination, when failure is caused by remnants of reactants used in bare board manufacturing (flame-proofing agents, copper plating deposits, etchants, cleaners, fluxes, etc.) or the PCB assembly process (fluxes, solder paste, "fingerprints"). Manufacturing defects also contribute to failures in electronic assemblies: loose solder, fillet lift, black pad, non-wets, oversized solder, insufficient solder, voids, cracks, etc.

4 Machine Vision

This section introduces the essentials of machine vision. A brief overview of theory and applications is provided to serve as an introduction to the field.

Machine vision tasks include methods for acquiring, processing, analyzing and understanding digital images. It is generally understood as a process of extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, for example, in the forms of decisions [11]. According to Bruce Batchelor and Frederick Waltz [12], Machine Vision should be distinguished from Computer Vision, with the former being a more applied area, critical to final solution cost, operation time, etc.

A typical machine visual system is designed to perform image acquisition and analysis, as well as the recognition of objects or object groups within an image. This process involves image sensing, representation of image data, and digitization. Subsequent analysis of the image may involve image segmentation (partitioning the image into meaningful regions), feature extraction (identifying the inherent characteristics or features of objects), or pattern classification (identifying an unknown object within an image as part of one particular group among several possible object groups). [13]

4.1 Components of Machine Vision System

A machine vision system is a complex assembly of lighting, optical, and electronic computer equipment [14]. The system should balance the following components: machine handling of the inspected object, lighting, optics, image sensing, and image processing. The interaction of components is displayed in Figure 2.

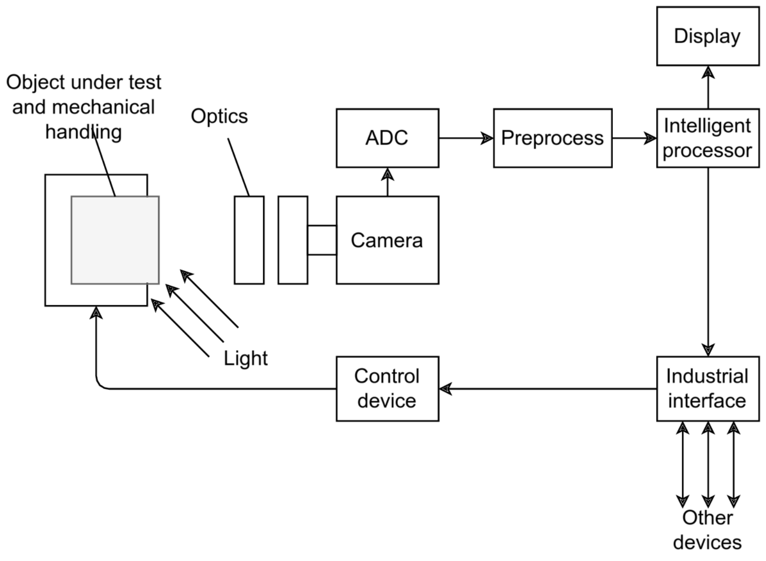


Fig. 2: Archetype of industrial machine vision system for inspection [14].

4.1.1 Lighting

The role of lighting and optics is to improve the quality of the image [14] [15]. The following light sources are commonly used: fluorescent, quartz halogen, LED, metal halide (mercury), and xenon. For small to medium-scale tasks, fluorescent, quartz-halogen, and LED are usually preferred, whereas metal halide and xenon are more typically used in large-scale applications where a very bright source is required. [16]

The relative positioning of the camera against the illumination source is called illumination technique. [15] describes over 100 illumination techniques for different applications. Commonly utilized techniques are the following: [16]

- **Backlighting**: Applied to get the silhouette of the object under inspection. A common application is the detection of holes, parts placement, and measurement.
- **Diffuse lighting (full bright field)**: Used on shiny or mixed reflectivity objects, where multi-directional light is needed (for example, to avoid shadows). The most common implementations are hemispherical/cylinder dome, on-axis, and flat diffuse light. Light should be placed in close proximity to the object.
- **Directional light (partial bright field)**: This is the most commonly used vision lighting technique [16]. This type of lighting has a defined direction and is used to generate a contrast image.
- **Dark field**: This type of lighting is characterized by a low or medium angle of light incidence, typically requiring close proximity.

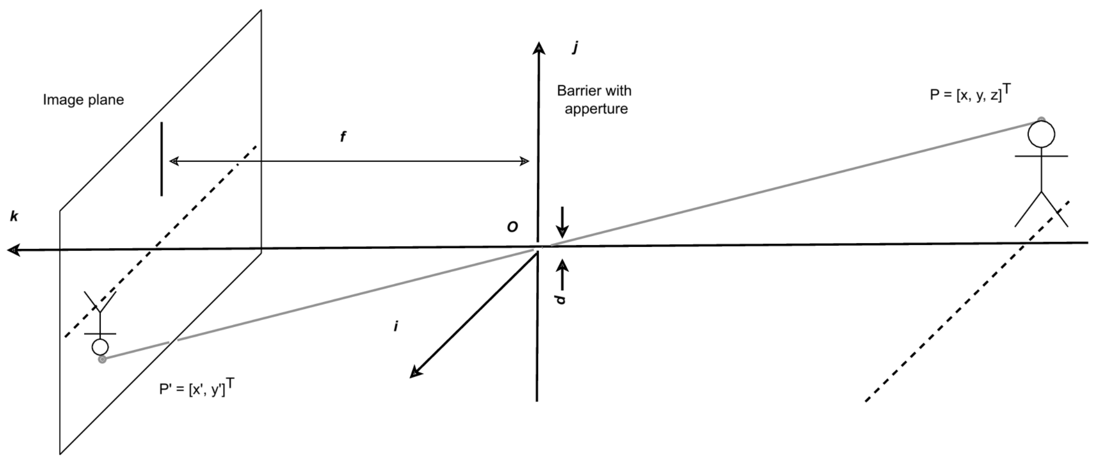


Fig. 3: Pinhole camera model, based on [18] [19].

4.1.2 Camera

In machine vision systems, the job of the image acquisition device is to receive a pattern of illumination from an optical system and convert this varying photon signal into a varying electrical signal that is a true representation of its spatial and temporal variations [15]. There are a lot of mechanisms to convert photon flux into an electrical signal. The earliest sensors were based on the photoelectric effect, which is still common in X-ray imaging. Modern sensors use photovoltaic material, which generates voltage across an in-fabricated diode when exposed to light (such materials are preferred in machine vision due to the linear relationship between input light and output voltage signal). Finally, the **image sensor** is made up of arrays of small structures designed to perform the basic functions of converting light to charge, holding the charge, and reading out the signal. There are two main sensor architectures available: CCD and CMOS. CCD, the earlier technology, after exposure is complete will transfer each pixel's charge sequentially to a common output structure, whereas in CMOS type of sensor, charge-to-voltage conversion takes place at each pixel. Many image defects happen at the sensor level, such as noise, image distortion, etc. [15] [17]

Image formation is traditionally described using camera modeling. The simplest camera model is known to be a pinhole model, which assumes that the perspective rays pass through an aperture at the front of the camera, as shown in Figure 3.

The pinhole camera model translates real world point (P) coordinate in image plane point (P') according to the following equation:

$$P' = [x', y']^T = [f\frac{x}{z}, f\frac{y}{z}] \tag{4}$$

where y', x' are coordinates in image plane, x, y, z are real world coordinates, f is an focal length. [19]

An improvement to camera can be made by placing lenses in the aperture to focus rays from each point in the scene onto the corresponding point in the image plane [20]. **Focal point** is a point where all rays parallel to the optical axis are focused and **focal length** is the distance between the focal point and the center of a lens [19]. The relation of *P'* to *P* is the same as for the simple pinhole model.

For a camera model with optics, the following properties are defined. **Depth of field (DoF)** is the width of that region on either side of the focal plane, where the image produced by an optical system remains in focus [15]. There is a trade-off between aperture size and depth of field: a smaller aperture provides more depth of field but admits less light, while a larger aperture admits more light but reduces the depth of field [20]. **Field of view (FoV)** is a cone in space that a vision sensor (eye or camera) can observe (described by two angles, measuring the horizontal and vertical limits of sight) [15].

Lastly, for a real camera, the following properties are defined. An **exposure** is the amount of light collected by the camera expressed in joules per square meter (depends on light intensity and duration of exposure) [15]. The **f-number** is a ratio of the focal length f to the diameter d of the aperture. The F-number on camera lenses is marked with numbers that are multiples of $\sqrt{2}$ because doubling the aperture area is equivalent to increasing the aperture diameter by $\sqrt{2}$.

4.1.3 Sensor Performance

The role of the camera is to provide information to the vision system. Different applications are sensitive to different aspects of the image, such as accurate color detection, fine detail level, etc. As of today, there is no settled standard to compare image sensors to each other. [15]

Generally, for digital image formation, **spatial sampling** and **quantization** are two concerns. Sampling is the process of converting from a continuous-parameter image to a discrete-parameter image by discretizing the spatial coordinate, while quantization is the process of converting from a continuous-amplitude image to an image that takes on only a finite number of different amplitude values. The combination of sampling and quantization is called **digitization**. The quality of the image is then determined by how closely spatially the pixels are located and how many levels are used in quantization. [15] [21]

Sampling has little impact when changes occur over many samples, but a large change in close samples is substantial and will result in image distortion. For a pattern of parallel black and white lines, the **spatial period** is defined as the distance between the center of one black line and the next. A combination of a white and black line is called a **line pair**. The **spatial frequency**, also referred to as spatial resolution, is then defined as the number of line pairs (lp) over a distance unit. For a sensor, the **limiting resolution** may be defined as the maximum spatial frequency that can be detected by this sensor. For an electronic camera using photodiodes, this limitation is given by the spatial distance between pixels. It is generally desired to minimize the number of used pixels while preserving most of the details in the image. The lowest sampling rate to preserve details is known as the **Nyquist frequency** and is determined as double the highest expected spatial frequency. Sampling at a lower rate will result in aliasing errors. An **alias** is a false lower frequency component that appears in sampled data acquired at too low a sampling rate. Alias overlays the true image information and cannot be removed by computation once it is generated. [15] [21] [20]

4.1.4 Image Processing

Digital image processing is a vast area of applied informatics. It includes an overwhelming amount of image processing techniques and approaches, from elementary pixel-by-pixel operations to modern deep learning-based techniques. It is out of the scope of this work to describe every aspect of image processing, so only basic techniques are the subject of this overview as an introduction to image processing. Notation follows [15]. Concrete methods used in the proposed system will be described separately.

A monochrome (greyscale) image is represented with a matrix $m \times n$, considering $0 \leq i \leq m-1$ and $0 \leq j \leq n-1$. The position (i, j) defines a pixel in the image. The pixels around a particular pixel are called the **neighbourhood**. Each pixel contains the value of intensity at this point. A modern image for machine vision usually has at least 512 pixels per side and uses 8-bit encoding to achieve 256 gray levels (0 for black, 255 for white). In a colorful image with an RGB color model, each point is represented with the following vector: $r(i,j), g(i,j), b(i,j)$, which define intensity for each channel.

Monadic, pixel-by-pixel operations assign a value to a pixel $c(i,j)$ of resulting image based on a single pixel $a(i,j)$ in source image. A common example is a **threshold** operation, defined as follows:

$$c(i,j) = \begin{cases} W & k_1 \leq a(i,j) < k_2 \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

where W denotes value for white.

Dyadic, pixel-by-pixel operations assign a value to a pixel $c(i,j)$ of the resulting image based on pixel values from two source images $a(i,j)$ and $b(i,j)$. Simple examples of such operation include addition, subtraction, multiplication, etc.

Local operations map pixels with neighborhoods from the source image to the resulting image. An important subset of such operations are **linear local operations**, which perform a weighted sum with a **linear local operator (weight matrix)**. This includes low-pass (blur) and high-pass (subtraction of a blurred image from the original image) filters. **Nonlinear local operations** include local min/max, median filter, edge detectors (with variations), and others.

A set of operations is defined for binary images. Basic operations for two binary images are inversion, logical AND, OR, and XOR. Using **dilation**, it is possible to expand white areas in the image, while **erosion** is used to shrink them. Consequent erosion and dilation are called the **opening** operation (**closing** is dilation followed by erosion). Other available operations include skeletonization, perimeter detection, edge smoothing, etc.

More complex techniques are used for shape detection (Hough transformation), spectrum analysis (discrete Fourier transformation), texture analysis (autocorrelation, histogram features, etc.).

4.2 Industrial Machine Vision

The term "industrial" for machine vision applications is used to emphasize the increased requirements for performance and stability. It is a commonly known fact that the industrial environment is demanding, and hardware will typically be used non-stop to achieve better production results. The following text aims to provide an overview of commonly used tools for industrial applications.

4.2.1 Common Hardware

According to the industrial machine vision market analysis for 2024 [22], the following companies are the major players: Basler, Teledyne, FLIR Systems Inc, Jai, Cognex, and others.

Modern cameras support a wide resolution range from 0.31 Mpx (VGA) to 29 Mpx. The number of pixels per side may vary from manufacturer to manufacturer. Common frame rate values range from 3 fps up to 500 fps for high-speed cameras (the frame rate may be increased by reducing the region of interest). Most cameras are manufactured to capture a range of the visible spectrum, from 380 to 780 nm (labelled VIS for visible), while there are sensors that are able to capture different wavelengths. [23]

Basic types of interfaces for connecting cameras to a computer are: USB 2.0 and 3.0, Gigabit Ethernet (GigE Vision), and Camera Link. Table 2 summarizes interface properties.

Label	**USB 2.0**	**USB 3.0 (USBVision)**	**Gigabit Ethernet (GigE Vision)**	**Camera Link**
Maximum transmission speed	480 Mb/s	4800 Mb/s	1000 Mb/s	2000 Mb/s (up to 5 Gbps)
Bandwidth	45 MB/s	600 MB/s	120 MB/s (basic bus) 255 MB/s (full)	680 MB/s
Maximum cable length	5 m	5 m	100 m	10 m
Maximum length with amplification	30 m	30 m	unlimited	30 m
Camera power supply	up to 0.5 A, 5V, via USB	up to 0.9 A, 5V, via USB	(except PoE)	(except POCL)
Delay of command (computer-to-camera)			350-500 µs	600-900 µs
Delay of command (camera-to-computer)			13-14 ms	13-14 ms
Computer processor load	average (5 to 10%)	very low (less than 2%)	average (5 to 10%)	very low (less than 1%)
Connection of multiple cameras	yes, via HUB	theoretically up to 63 devices	yes	no
Wireless support	no	no	yes	no
Relative price	low	low	low	high

Tab. 2: Summary of communication interface properties [23] (reduced).

The choice of computer processor is not critical. Almost any modern personal computer will be suitable with appropriate package for plant use to provide sufficient protection and maintenance options. [15]

4.2.2 Common Software

Early machine vision systems were built with low-level programming. Such systems were based on simple frame grabbers providing low-level interface capabilities with other system components, basic user interface (UI), and analytics algorithms. This approach experienced difficulties in system integration and maintenance. Eventually, machine vision inspection systems became more modular, which provided more abstract capabilities for system development and maintenance, leading to higher robustness. [24]

As for operational systems, common commercial OS are often chosen (in case of modular systems, smart cameras and embedded systems are different story). Windows and UNIX based systems (Linux) are considered the main alternatives. Windows is often preferred to achieve labor saving application development with maximum portability. However Linux is becoming a standard for highly customized or cost critical design.

If high customization of machine vision systems is not a concern, it is possible to utilize commercially available image processing environments. To be suitable for industrial inspection, such environments must contain algorithms for edge and line detection, image enhancement, illumination correction, geometry transforms, Region of Interest (RoI) selection, object recognition, feature selection, and classification. Examples of such systems include: Khoros, SCIL-Image, LeadTools, IPL Lib, Sherlock32/MVTools, Image-Pro Plus, OPTIMAS, various fuzzy systems, MatLab [25], NeatVision [15], and others. Very few of them are provided as open source and often require costly licensing. As for programming languages, C and C++ are often used, and even Prolog [15] [12] [24].

4.2.3 Integration Into a Factory Environment

A factory floor inspection system (**target inspection system** [12]) must be able to communicate with shop-floor personnel and control actuators. The minimum requirement is that the system should reassure workers that it is operating properly. The main function of such a system may be as simple as an accept/reject gate, but even in this case, Human-Machine Interface (HMI) is a good practice. In order to control peripherals, the system is expected to use standard interfaces, such as a serial port for example. [12]

5 Literature Overview

This section aims to present a comprehensive review of the state-of-the-art algorithms used in the modern AOI industry for 2D systems. Other popular approaches are described in market overview, chapter 6 (3D systems and proprietary inspection algorithms in particular). The algorithms chosen for implementation will be discussed in detail in other sections.

5.1 General Inspection

To identify the most suitable detection algorithm for a defined range of problems, a literature review was conducted on general defects, such as missing components or misaligned components. Within the realm of PCB inspection literature, several key research areas have emerged: bare PCB board inspection, component defect inspection, and solder joint inspection. This section concentrates primarily on the literature and algorithms pertaining to component defects.

The literature overview is based on research of Qin Ling et al. in year 2023 [26], which comparatively studied various defect detection methods proposed between 2000 and 2022. Their research focused on traditional image-processing, machine learning-based, and deep learning-based techniques. For a more general survey of AOI systems, Abdalrahman M. Abu Ebayyeh and Alireza Mousavi [3] provide an in-depth overview of optical inspection in the electronics industry. Their article covers a wide range of commonly inspected electronic components, including semiconductor wafers, flat panel displays, printed circuit boards, and light-emitting diodes.

5.1.1 Image Processing-Based Methods

These methods leverage classical image processing algorithms to achieve defect detection. A basic example involves combining image binarization using a threshold with a subsequent XOR operation to identify potential discrepancies. However, more sophisticated approaches are necessary to ensure stable results under variable conditions, such as fluctuating illumination.

Hassanin et al. [27] proposed an inspection system that utilizes the speeded-up robust features (SURF) algorithm for image alignment (image registration in the paper's terminology) [27]. This is followed by image segmentation using Otsu's thresholding method combined with morphological operations to differentiate foreground pixels from a template for missing component detection. Their system reportedly achieves an inspection time of 3-4 seconds, excluding defect type identification, which requires an additional 2-3 seconds [27]. However, as noted in Qin

Ling et al. [26], this method might be inadequate for PCBs with high component density.

Sundaraj [28] proposed a background subtraction method for PCB inspection that relies on background modeling using a Mixture of Gaussian (MoG) approach. The background is modeled from 100 frames of the PCB (approximately 3-4 seconds during initialization) and then subtracted from the inspected image (which requires image alignment). While this method is effective for detecting missing or misaligned components, it has difficulty processing components with the same color as the PCB itself. The authors achieved over 90% accuracy for foreground detection [26]. Qin Ling et al. [26] categorized this approach as time-consuming and unsuitable for real-time applications, despite the authors' claim of achieving an operational rate of 20 images per second for a 640 × 480 pixel image [28].

Several studies have explored approaches based on normalized cross-correlation (NCC) for PCB inspection. The original NCC method is described in [29], while an improved version, referred to as INCC (improved normalized cross-correlation), is presented in [30]. The INCC method boasts reduced computation time and robustness to varying illumination conditions. While the authors of [30] successfully applied INCC for missing IC detection, the effectiveness for smaller components remains uninvestigated. Additionally, as noted in Qin Ling et al. [26], the processing time of several seconds per frame renders this method unsuitable for real-time applications.

Han-Jin Cho and Tae-Hyoung Park [31] proposed a defect detection technique based on wavelet transformation. The authors implemented several iterations of discrete wavelet transform (DWT) followed by cross-correlation to assess the matching rate between a test image and a template. Their method achieved an accuracy of 86.6% for normally illuminated images with a processing time of 44 milliseconds for a 256 × 256 pixel image [31].

5.1.2 Machine Learning-Based Methods

These methods leverage various classifiers, including universal approximators like neural networks, to achieve superior defect detection capabilities compared to traditional image processing approaches.

One such approach, proposed by Ahmed et al. [32], combines wavelet transformation (DWT) with a multi-layer perceptron classifier for defect detection. However, this method necessitates the creation of a comprehensive database containing all known components and potential defects to train the classifier effectively. Each component requires a dedicated classifier trained using synthetic data generated from the database. Notably, performance metrics for this approach are not reported in either the original work by Ahmed et al. [32] or the comparative study by Qin Ling et al. [26].

Crispin et al. [33] introduced a variant of the template matching approach for component presence detection. Their method employs a modified genetic algorithm (GA) to locate a generalized template of a component on the PCB. This modification allows the GA to detect multiple components simultaneously. Normalized cross-correlation serves as the fitness function in this scenario.

Li et al. [34] employed depth images with a semantic segmentation method and a random forest pixel classifier for PCB inspection. However, this approach exhibited significantly lower accuracy (83.6%) on real PCBs compared to synthetic data [34]. Additionally, it requires specialized equipment and calibration procedures, posing practical limitations.

5.1.3 Deep Learning-Based Methods

There are several papers available that aim to apply deep learning (deep neural networks) for PCB inspection needs. It is worth noting that many studies related to deep learning applications for PCB defect detection are not included in this review as they are not relevant (such as solder joint inspection and bare board inspection).

A common issue when dealing with deep learning is data availability. During the preparation of this review, a few publicly available annotated datasets were found. These include the FPIC dataset [35] with 6.2K annotated images for component classification, the PCB Defects dataset [36] with 10K images of bare PCB defects for detection, the HRIPCB dataset [37] with images of bare PCB defects for detection, and others. However, a sufficient dataset that would be suitable for the desired inspection flow was not found (available datasets are not labeled for missing component detection, misalignment, text issues, etc).

Shen et al. [38] proposed a lightweight PCB component detection network (LD-PCB) achieving 85% mAP within 0.0036s per image, where mAP stands for mean average precision. They also introduced a lightweight character recognition network (CR-PCB) with 96.7% mAP within 0.018s. The entire system was reported to achieve 95% accuracy in defect detection. The inspection process generally consists of component detection and identification, enabling the detection of missing and incorrect components.

The PCB component classification task was addressed by Dhruv Makwana et al. [39]. They proposed a PCBSegClassNet lightweight network for component classification (25 classes), trained on the FPIC dataset. The paper also includes a comparison of the proposed model with eight state-of-the-art classification networks, namely ResNet50 (He et al., 2016), MobileNetV2 (Sandler et al., 2018), EfficientNetB0 (Tan & Le, 2019), DenseNet121 (Huang et al., 2017), DInPNet (Mantravadi et al., 2023), PVT-small (W. Wang et al., 2021), PVTV2-B0 (W. Wang et al., 2022), and Swin-Transformer (Liu et al., 2021).

5.1.4 Summary

Table 3 summarizes the discussed methods (based on [26]). As of today, an ultimate solution for component-related defects does not seem to have been found, as all methods have considerable disadvantages.

5.2 Text Recognition

Text recognition is a common task used to validate whether the required component was mounted by performing component identification. This task is often separated from the general inspection process because it is not specific to the PCB industry and is generally referred to as OCR, which stands for Optical Character Recognition.

Anitha D. B. and Mahesh Rao [40] utilized OCR for component value identification. The described inspection system is built using LabView OCR NI module, and a training process is required to implement OCR. The authors reported 3 ms for value detection, but the overall inspection time of the designed system (including other tests) was reported to be 14-29 seconds when using OCR.

Wei Li et al. [41] studied OCR for PCB recycling purposes, focusing on image binarization and final text recognition. As part of their research, the authors performed a comparison of available open-source OCR engines. Among the tested OCR engines, Tesseract and Cuneiform provided more solid performance compared to GOCR and OCRAD. The authors also demonstrated that the quality of text recognition can be significantly improved if the binarization approach takes into account the technical limitations of OCR software.

Yuzo Iano et al. [25] developed an inspection system using Matlab OCR tools, with a reported time per image above 285 ms.

Sumyung Gang et al. [42] investigated deep learning capabilities for character recognition of components mounted on PCBs. The authors proposed an EfficientNet model for character recognition, trained on 8,000 to 12,000 images per class, achieving high accuracy in detection (99.065% accuracy for the B7 model).

Ref.	Concept proposed	Criterion	Result	Advantages	Disadvantages
[28]	Background subtraction	Detection accuracy	> 90%	- Large region monitoring - Increased handling capacity	- Fails to detect the component that has the same colour as the background - Same size and orientation of reference and inspected image is required - Time consuming
[30]	INCC	Detection accuracy	> 96%	- Less false alarms - 4x faster then conventional NCC - More robust to illumination changes	- Time consuming - Require same size and orientation of tested image
[27]	SURF technology	AROC curve	> 95%	- Detection regardless of PCB orientation	- Time consuming - Depends on local pixel information
[32]	Combined DWT and MLP	NA	NA	- Common light source	- Require template image - Database knowledge base should be constructed prior
[33]	Combined GA and template matching	NA	NA	- Successfully detect resistor presence	- Require template - Time consuming
[34]	Random forest pixel classifier	Detection accuracy	98% or 83%	- High recognition rate for synthetic images	- Poor performance on real images - No defect classification mechanism - Time consuming - Sensitive to parameters selection
[38]	LD-PCB and CR-PCB to detect component defects	Detection accuracy	95%	- High accuracy for component defects	- Template marching or image comparison was required - Three-stage method - Not real time

Tab. 3: Summary of proposed inspection approaches.

6 Market Overview

The automated optical control systems market is experiencing demand growth. In 2020, the industry was valued at $1 trillion [3]. The market is divided into inline and offline AOI. Inline AOI systems are directly integrated into production lines, while offline systems require additional PCB manipulations (thus not allowing for continuous production flow). The inline AOI segment is leading in the market. Systems are further divided into 2D and 3D. The 2D approach is cheaper and simpler for implementation. [43]

Short list of key companies in the AOI market [43]:

1. Koh Young Technology, South Korea (established 2002)
2. Test Research, Inc., Taiwan (established 1989)
3. Omron, Japan (established 1933)
4. MIRTEC Co., Ltd., South Korea (established 2000)
5. Viscom, Germany (established 1984)

6.1 Industrial Systems Overview

In this section, a brief overview of commercial industrial systems is presented. The overview is based on systems available on the PCB Directory website [44]. PCB Directory is the largest online directory of PCB Manufacturers and Fabricators, with over 1,000 manufacturers from all over the world [45]. The overview is focused on applied inspection approaches. The presented information is publicly available in product data sheets on the PCB Directory website.

3D systems are mostly based on a multiple-view 3D image reconstruction approach [46]. This approach allows 3D object reconstruction by combining images from multiple cameras located around the object of interest. This approach allows for 3D measurements and shape extraction, as well as controls based on tolerances. The exact inspection algorithms are not known in most commercial systems of this type. Examples of systems include the Viscom S3088 ultra series [47], and the XCeed 3D AOI [48]. Some systems combine 3D and 2D inspection approaches, such as the Juki RV series [49].

Image processing is a family of deterministic algorithms for image inspection. Such algorithms are often based on histogram properties of the image, color analysis, contrast enhancement techniques, pixel neighborhood processing, etc. Usually, manufacturers offer a complex system capable of detecting most known PCB defect types.

Most 2D AOI systems use RGB lighting for solder joint defect analysis. The basic idea of this approach is to use three sources of light with different colors (red, green, blue) under different angles to encode height information and then process this information separately for a few areas of interest [50]. Examples of such systems include the Omron VT-S730 [51], the 2D AOI-V5000 [52], and the Acrotech TR 7700 SIII DT [53].

A few cases of deep learning utilization were found. The AIS20X Series from Maker-ray company uses CNN for PCB component classification [54] (Nvidia GPU used). The Jaguar Off-line AOI Inspection Machine A1000 [55] also claims to use AI in the inspection process.

6.2 Summary

The following list summarizes methods that were reported to be used by related manufacturers:

- 3D image [47], [48], [49]
- RGB light-based inspection [50], [51], [52], [53], [54], [55]
- Histogram-based methods [56], [53], [57], [58]
- Image contrast algorithms [59], [57], [58], [54]
- Image similarity [59], [60]
- Color linear/non-linear analysis [59], [60], [61], [62], [57], [58]
- Black-white ratio analysis [61], [62]
- Statistical analysis [61]
- OCR, OCV [59], [60], [61], [57], [54]
- Multistage analysis [53]
- Multiple simple control mechanisms [62]
- Pattern recognition [62], [54], [55]
- Brightness level analysis [62]
- Machine Learning [57], [58]
- CNN [54]
- AI [49], [55]

7 System Requirements

This section aims to describe the required features of the developed system. The section also defines the foundation for the system's performance assertion (acceptance criteria). Before specifying requirements, it is necessary to define formal roles for parties related to the project development and consumption. The person responsible for the system development is referred to as the **developer**. The person or organization intended to use the proposed system is referred to as the **end consumer**. The system under development is referred to as **the system** or **the project**. The system includes hardware and software required to fulfill defined goals and other related materials and sources needed to use the system. The system components will be closely described in other sections. Requirements were defined by the end consumer during the project preparation phase.

7.1 Inspection Process

The system is required to perform inspections on printed circuit boards. The inspection process entails the ability to detect and react to a defined set of PCB defects. Inspection should be performed using 2D images of the PCB (gray or colorful). Inspection should be conducted against a **template (golden) image** of the **reference PCB** provided by the end consumer during the configuration process via the graphical user interface (GUI). The system should be able to handle the following PCB defects (although it may handle other defects as well):

- **Missing component**: The defect, for the purposes of this paper, is defined as the case when a component (not closely specified) may not be observed inside the defined area of the PCB under test, but may be observed inside the same area on the reference board. Closer definition may be specified for each particular detection algorithm.
- **Print quality issue**: The defect is defined based on the percentage difference from the template image. The system should be able to compare print quality against the template.
- **Capacitor wrong polarity**: The defect is defined for an electrolytic capacitor with color polarity marking. The inspection should identify the opposite capacitor polarity than desired.

The system is desired to support **text recognition**, which is defined as the general ability to recognize text on the board. Text properties are not specified in advance. This type of inspection should provide recognized text as a result.

Inspection should be performed in an automated manner without operator assistance during the inspection process. The PCB is not closely specified in advance. The system should provide a configuration interface for fine-tuning. Configuration should be performed prior to inspection by the end consumer. Detected defects should be properly logged. The log should include defect type, defect location, and an image of the detected defect. Detected defects should be immediately reported to the operator and to upstream factory systems.

7.2 Operational Environment

The system should be designed to be integrated into the production line and should include all required components for operation. Image acquisition should be done by the inspection system's own means. Communication with upstream systems should be realized via digital I/O pins. The communication protocol should be defined by the inspection system so that the upstream system is able to trigger inspection when a new PCB is ready for inspection and obtain inspection results in the form of general conformity (indicating any failed tests).

The system is expected to operate in inline mode without internet access. Deployment should be conducted by the developer or another person with the required skills. It is expected that the operator will have access to the system on-site, near the place where the inspection is conducted. Input to the system should be controlled with a mouse and keyboard (potentially touch-screen).

7.3 Access Management

As user interaction with the system is required, the system should be protected from unauthorized access (no anonymous access). The system should also provide role-based user management to prevent unauthorized access to process settings. The role should be defined for an *operator* to monitor the inspection process without the ability to change process settings. Another role should be defined for the person responsible for the inspection process configuration (*administrator*), and the administrator should be able to perform the functions of the operator role and general system configuration.

7.4 Performance Requirements

The system is required to perform inspection within 3-5 seconds after the trigger signal is detected. This time includes image acquisition, image preparation, inspection, and reporting. Performance evaluation of the system is required.

7.5 Other

Software architecture and the deployment process are not closely specified; however, the system should be portable and extendable. The target platform for basic inspection tasks is a common personal computer with an optional graphics card. An industrial camera is required.

8 Open AOI

This section presents the proposed system. The system overview will be provided with specific implementation details. The experimental setup will be covered in a separate section. Related design concepts are also described to provide design context.

8.1 Design Context

Software development and IT service delivery have come a long way from early paper-drilled programs to modern cloud-based solutions. Today, a few common approaches to software development and deployment have emerged, each with its own advantages and issues.

The basic form of service delivery is the development and deployment of custom standalone applications. This type of program does not require any external modules, library functions, or other programs and is executed directly on the consumer's hardware. This type of program may not require installation (portable standalone application). This approach is one of the earliest and, as mentioned in Chapter 4, was in fact used for Machine Vision applications. Applications are developed with C++, C#, Java, and other languages. The approach is capable of handling all required functionality while providing solid performance with high processing speed with code close to the hardware. Unfortunately, such systems are hard to develop and maintain with a growing code base, increasing internal dependencies, and compilation time.

In order to reduce system complexity, maintenance and development efforts, and to increase overall system stability, a tendency to utilize the isolation of individual service parts has emerged in modern design approaches. A common example is microservices architecture. Microservices architecture is a pattern that arranges an application as a collection of loosely coupled, fine-grained services, communicating through lightweight protocols. A common choice in terms of communication protocol is a TCP/IP protocol with JSON or XML as the data format. Service isolation is beneficial for testing, load leverage, system stability (if a service fails, requests will be forwarded to another service and users will not notice any downtime), and many other important factors. The cost of such an approach is the time delay caused by inter-service communication, which may be an issue with real-time or close-to-real-time applications. Another downside is the requirement for an application interface (API) to allow service communication. Basic microservices architecture may be as simple as two operating system processes communicating with each other via API. A more complex example of this approach could be Microsoft Azure functions, which is a serverless computing platform.

For robotic applications, microservice architecture was adopted with the ROS system, which utilizes the isolation of application parts into nodes and provides a message exchange system to allow nodes communication. This approach allows encapsulating complex computations into simple yet solid interfaces. The latest ROS systems provide QoS control capabilities; more on that later.

Another common issue with IT service delivery is application hosting and deployment. The application has traditionally been hosted on the consumer's infrastructure (computer). For large and complex systems with distributed functionality and multiple dependencies, such a hosting model is not feasible, as complex systems often require more than simple installation (resource maintenance, monitoring, software and hardware updates, etc.). In case the system should be deployed to the consumer's environment, it is advisable to use containerization (Docker), which allows completely isolating the application execution environment and providing all required dependencies without concerns about the consumer's hosting environment (operating system, installed libraries, etc.). After the application has been isolated, deployment to the execution host and service orchestration could be leveraged to systems like Kubernetes. Kubernetes is designed to allow services, for example, to scale on demand (utilize more computational power). Finally, as more computationally demanding applications are developed today, it is common to rent computing resources (hardware) from computing providers and deploy solutions in the form of cloud-based applications. Such an approach is known as hardware-as-a-service and comes in handy when large computational resources are required (many computational units, graphical cards, etc.).

It is absolutely possible to build an AOI system with any of the described approaches. However, the chosen approach will have an impact on service quality and the level of flexibility for consumer customization. It will also define the scope of applications where the system may be used.

An important concept for this work is the concept of **open-source**. The open-source software movement, according to Wikipedia, branches from the free-software movement, which originates in the 80s, although code sharing practices were developed even earlier in the 50s. The movement supports the use of open-source licenses for software distribution, and such software is known as open-source software. The license usually grants users the rights to use, study, change, and distribute the software and its source code.

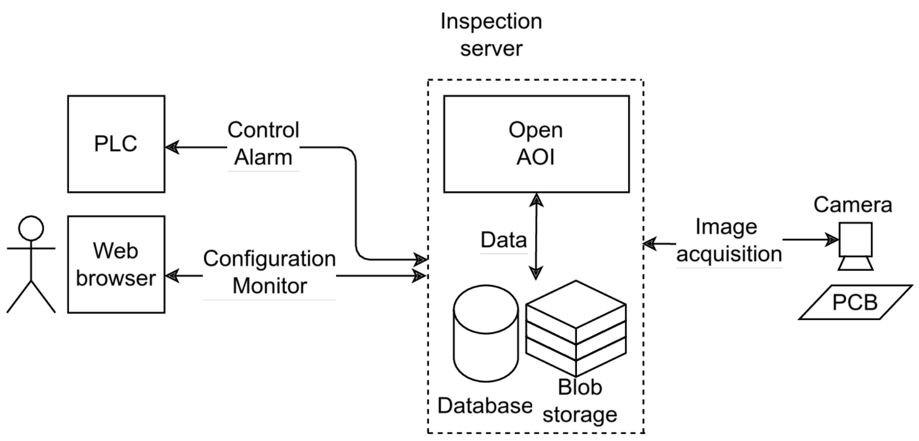


Fig. 4: Open AOI system concept.

8.2 Proposed System

Open AOI stands for **Open Automated Optical Inspection**. From a practical standpoint, Open AOI is a framework that aims to simplify the development and deployment of optical control systems. The word *open* in the system name refers to the open-source nature of the system. The system is designed with open-source software and is provided under the MIT license as an open-source project.

Open AOI brings together hardware and software to cover most common optical inspection tasks. By default, the system also provides a few inspection modules that cover requirements defined earlier in this work. Figure 4 shows the conceptual system design. The system is designed to be hosted on a single computer, while distributed hosting of different system parts is also possible (including distributed hosting of ROS nodes, which will be discussed later). While the system currently provides only PCB inspection capabilities, it is not restricted by design to PCB-only applications and may be generally applied for any type of optical control.

8.2.1 System Components

The system consists of several individual components:

- **Open AOI portal**: This is the main user interface for interaction with the system. The portal is designed as a web application and will be discussed in the following sections. Technically, the portal is a ROS node; however, this component is a separate logical part of the system.
- **Open AOI ROS packages**: The main resources for inspection handling are defined as ROS packages. ROS nodes communicate with hardware, conduct the inspection loop, identify PCBs, and serve some other functions. There is also a ROS package that provides wrappers for Open AOI database and blob storage.

- **MySQL server**: The Open AOI system stores structured data such as users, logs, and hardware meta-information in its own MySQL instance. The Adminer interface is available as part of the deployment suite for simplified external data management. The database may be hosted on a separate host if required.
- **Minio blob storage**: In order to store collected images during the inspection process, blob storage is utilized. Images and user-defined modules are stored as bytes in self-hosted blob storage. This component may be hosted on a separate host if required.

A camera and hosting computer were not included in the list of components as they are expected to be configurable elements of the system and do not require any specification at this point (the designed system is not dependent on the hosting environment or the used camera). Figure 5 describes the general interaction of components. It also provides a more detailed insight into the ROS part of the system.

The system is modular by design. Inspection logic is realized as a collection of ROS nodes that communicate with a mediator node (utilizing the mediator pattern). The following ROS services are available:

- **GPIO interface**. This node provides communication capabilities to external systems. It monitors general-purpose I/O pins and issue mediator commands. It also acts as a trigger for the inspection process. The node relies on the Raspberry Pi GPIO library (when deployed to Raspberry Pi).
- **Web interface**. This node hosts the web application. The web application is built with the NiceGUI framework and communicates directly with the mediator node. This node is also the only node that is allowed to communicate with other nodes directly (without moderation) in order to reduce communication delay for a better user experience.
- **GPIO interface**. This node is responsible for GPIO handling. Node realize communication with external systems via I/O pins.
- **Mediator**. This node acts as the supervisor for other nodes. The main inspection flow is defined here. The node should process external triggers, direct the inspection process, store inspection results, etc.
- **Image acquisition**. This node provides the camera interface to the system. It encapsulates the camera setup and control process.
- **Product identification**. This node is designed to identify barcodes in the image.
- **Inspection execution**. This node is designed to apply inspection modules with given parameters on the provided test image and template.

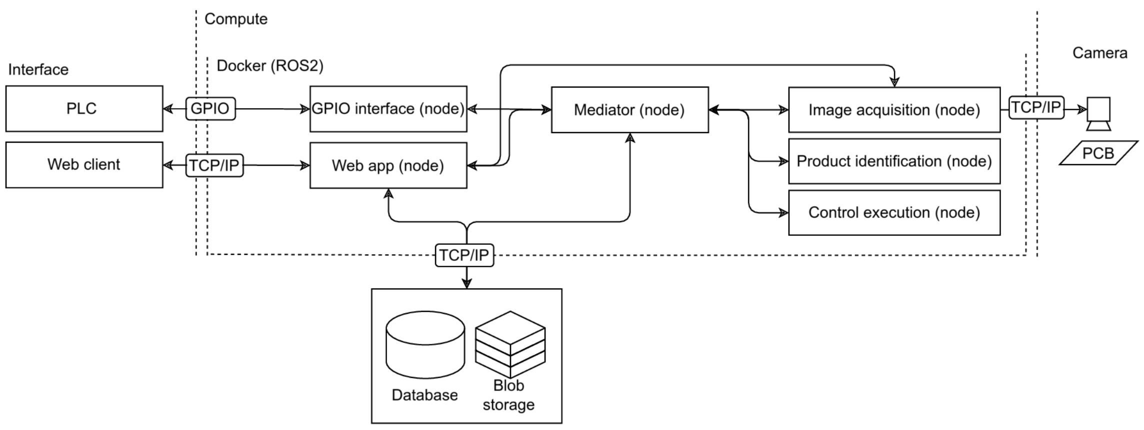


Fig. 5: Open AOI system components.

8.2.2 Components Justification

Open AOI may seem over-engineered at first glance. Common optical inspection tasks may be solved with the use of a simple script looped to acquire images and test them with predefined algorithms without ROS utilization, databases, GUI, etc. Even though such an approach cuts most of the short-term costs related to development, it will cause higher losses in the future. Such "simple" solutions tend to become messy and lose maintainability when extensions are implemented, even with the best efforts to preserve the code base. The absence of a proper user interface and clean data management policies will result in data loss, system inconsistencies, security issues, stability issues, and so on. The last thing which is of great concern from a technical standpoint is performance and portability. With low requirements, none of the mentioned is a concern; however, when requirements such as close to real-time application, increased computational power requirement may potentially raise, a more sophisticated system design is generally required.

The following text justifies the used components and describes potential use cases and improvements of the proposed system.

ROS2

According to ros.org [63], ROS (Robot Operating System) is an open-source software development kit for robotics applications. ROS is generally used for embedded systems and is designed to handle control of self-driving autonomous cars, robot hands, etc. The Open AOI utilizes ROS2 (developed with the Foxy distribution), which is a second revision of ROS firstly released in 2017. ROS2 offers considerable improvements over ROS1 [64].

ROS is widespread and well-known in open-source robotics. In combination with high modularity and the ability for nodes to communicate over the network, ROS provides a great compatibility background for Open AOI, allowing for native integration with robot hands, manipulators, AGVs, etc. Cross-network communication also allows for the separation of computationally intensive parts of Open AOI (like text recognition) to isolated computing environments with specialized hardware (GPU).

Both ROS1 and ROS2 support both Python and C++ languages. This allows Open AOI to utilize the speed of C++ if ever required. In the current implementation of Open AOI, C++ is not used.

ROS2 is built with native support for real-time applications. Built upon DDS (Data Distribution Service), which is suitable for real-time applications because of its various transport configurations (QoS) [64]. For Open AOI, this means the potential to be used in real-time or close to real-time applications with little to no modifications to the core of the system. [64] provides proof of concept for DDS utilization and metrics for internode communication in ROS1 compared to ROS2.

Docker
The Docker is another important component of Open AOI. Docker is used to isolate Open AOI from the execution environment and provide all required dependencies. Docker utilization allows simple deployment of the system with minimal efforts. Dependencies and required services are defined as a text file (docker-compose.yml) and are installed during installation or before installation (in case of prebuilt images). Docker is an optional dependency of this project, and direct manual installation to the target machine is also possible.

8.3 Design Details

Development of a complex system usually starts from high-level abstractions to identify basic entities and their relationships with each other within the scope of the solved problem. The following text provides a detailed description of the internal structure of the Open AOI system.

8.3.1 Role Model

To implement proper data protection, a role-based approach has been adopted. A role is a collection of access flags (privileges), which are checked on every access attempt. Two main roles are defined as **Operator** and **Administrator**. Each user should be registered in the system as an **Accessor** and should have an assigned role. An Accessor instance has a username and password. This way, anonymous access is prohibited as no anonymous role is defined in the system. The role-based model is only valid for the Open AOI portal. Other parts of the system define independent access control mechanisms.

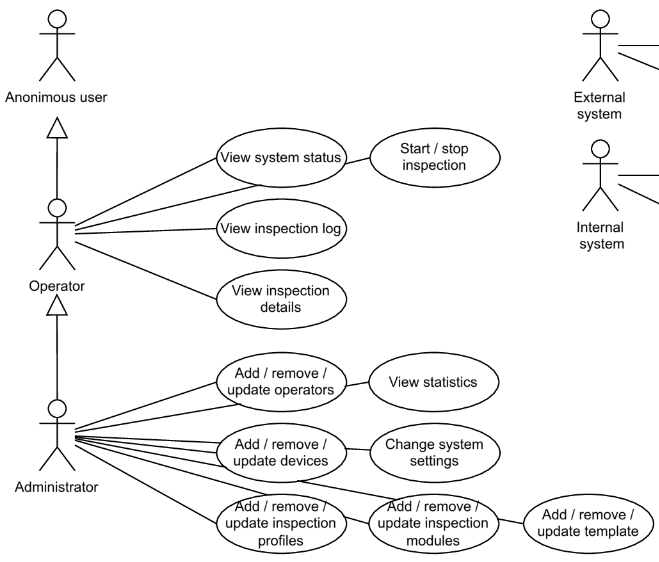


Fig. 6: Use-cases considered for implementation of Open AOI.

Other entities that may interact with the system, such as the upstream system and the system itself, do not have dedicated roles, and their actions are not tracked directly. This behavior is subject to change in the future as it poses certain security issues.

8.3.2 Use Cases

Based on the role model, Figure 6 describes the use cases that were considered for implementation.

The operator is granted permission to perform a set of harmless operations without access to potentially sensitive information, such as inspection configuration, statistics, devices, etc. The operator thus does not have permission to change the system's state. The only exception is for inspection flow control, as it may be required to intervene under certain conditions in the inspection flow.

The administrator is granted all possible permissions. The system may prevent some actions by default to prevent potential data loss or inconsistencies in inner data.

8.3.3 Domain Model

The domain model defines detailed interactions for internal entities while avoiding implementation details, such as exact properties, methods, class names, etc.

The domain model (Figure 7) specifies a few additional entities, namely:

- **Camera**. The system allows the definition of multiple cameras (each with its own title and IP address).
- **Template**. Templates hold images to perform inspections against.

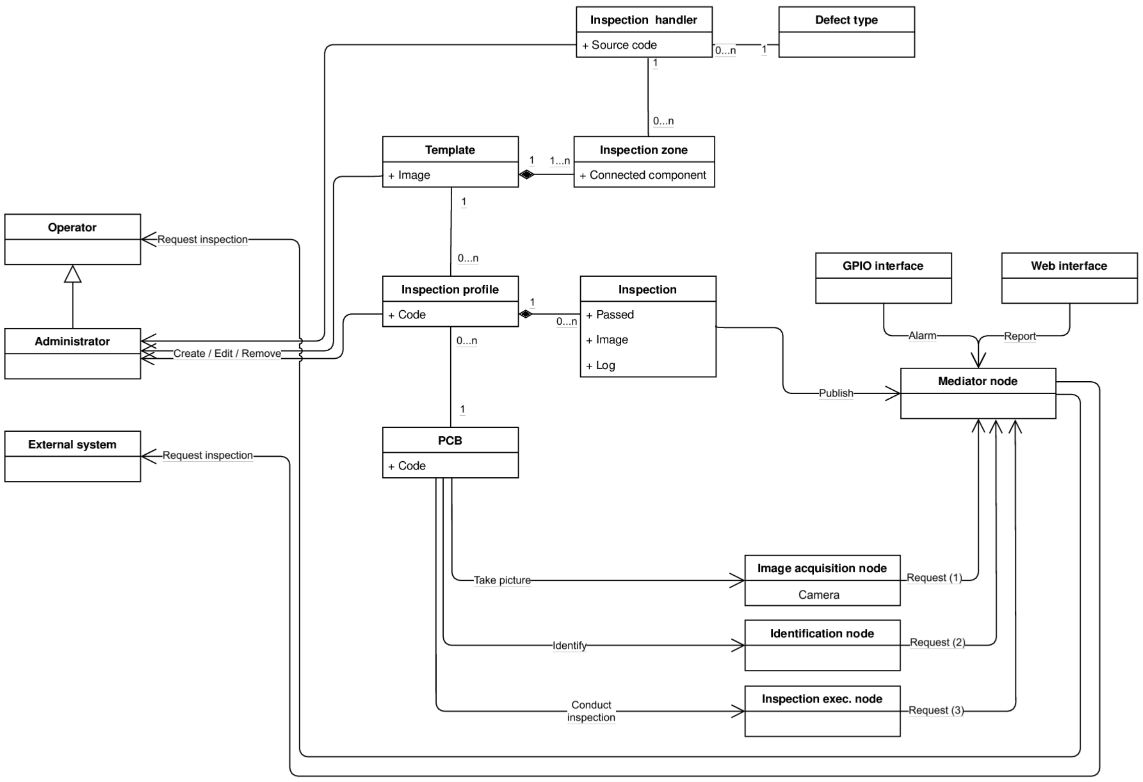


Fig. 7: Open AOI domain model.

- **Inspection profile**. Inspection profiles bind together inspection templates and inspected products by identification code (barcode) value. The inspection profile will be looked up when the product under test is identified.
- **Inspection zone**. Inspection zones are rectangular areas defined on templates. They specify specific areas to apply inspection handlers to in order to identify the presence or absence of related defect types.
- **Inspection handler**. Inspection handlers (also known as modules or custom code) provide concrete inspection algorithms for the inspection loop. Each inspection handler should be related to a single defect type that it detects.
- **Defect type**. Defect types describe expected defects, for example, missing components or print issues.
- **Inspection**. Inspection records are generated for every conducted inspection and contain inspection results and some other related information (including captured test image).

Other entities have already been introduced (services, roles, and interfaces). The PCB entity is an abstract PCB under inspection to better illustrate inner interactions. The role of the mediator node as the system's central element may be observed here.

9 System Components Implementation

Open AOI is implemented as a collection of ROS packages. There are a total of 5 packages developed, namely `open_aoi_core` (*core*), `open_aoi_interfaces` (*interfaces*), `open_aoi_portal` (*portal*), `open_aoi_services` (*services*), and `open_aoi_gpio` (*gpio*). The system is deployed as a set of Docker containers. The following text describes the deployment and each package in detail.

9.1 Docker Services

OpenAOI requires a database, a blob storage, and an Ubuntu-based operating system with ROS2 installation. To provide all required dependencies, Docker containerization was utilized. Each service, like the database and blob server, exposes ports for users to access them directly. For MySQL, an Adminer database manager has also been installed to simplify occasional manual interaction with the database (even though it is highly discouraged to make any manual changes in the system's database).

It is worth mentioning that ROS2 Foxy is currently deprecated as newer versions of ROS2 have been released. This decision was made due to compatibility issues with the developer's local environment. For future releases, the ROS version will be upgraded to the long-term support version. Incompatibility arises from the fact that ROS uses the system's global Python interpreter and does not allow the use of a virtual environment (technically it is possible, but no official way to do so was found). The system was developed under Ubuntu 20 on a personal computer with Python 3.8, which is compatible with ROS2 Foxy at maximum.

9.2 ROS Packages

9.2.1 Core Package

The core package is a Python-based package that contains the implementation of interfaces for communication with the MySQL database and the Minio blob server. This package also contains default content for the OpenAOI system and tools to deploy default content to related parts of the system during installation. Conceptually, the core should contain shared parts and should be imported into other packages.

The core defines database schemas with the SQLAlchemy Python package. Database schemas are created during system deployment. To manipulate database data, an ORM (Object-Relational Mapping) approach is utilized. This means that each object in the database, such as inspection zone, is mapped to an object in

Python, and changes in the Python object are reflected back to the database upon commit. Such an approach is advantageous in that it allows the implementation of additional methods to database entities. This is done with so-called mixins, which are classes in Python that add additional functions to objects by inheritance. To provide blob storage capabilities, blob storage mixins were developed for inspection handler entities and entities that store images (inspection, template). Mixins allow the isolation of functionality logically and the reuse of code where required by simply inheriting from the required mixin.

The core also defines default content (default modules, default defect types) that is deployed with the system. This content is related to end consumer requirements and provides capabilities to detect missing components, text quality, etc.

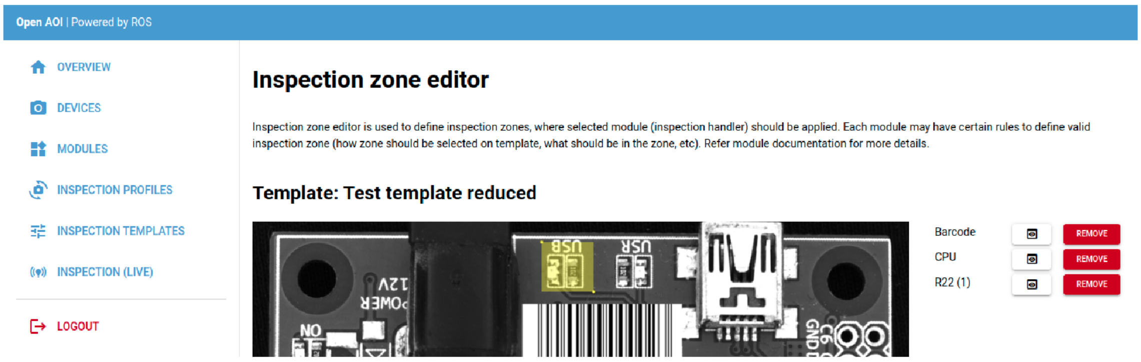
9.2.2 Interface Package

ROS2 nodes communicate via messages and service requests and responses. Each such object has its own structure, which is defined in ROS format in the interfaces package. Upon build, ROS translates them into Python code, and messages can be used with the ROS client library for communication (in the case of Python, it is `rclpy`). It is considered good practice to separate these definitions into a package. The package is fairly simple and defines all required objects to enable internode communication. The image is passed between nodes as a built-in ROS image type.

9.2.3 Portal Package

The OpenAOI user interface is a web application that is written with the NiceGUI framework and is hosted as a ROS node in order to directly communicate with the ROS system. Figure 8 shows page were user will configure templates for inspection. The original idea was to host the portal aside from ROS and to implement communication via a ROS bridge, which is a ROS package that contains a ROS websocket server and provides a Python SDK to communicate with this server. Such an approach would allow for even more isolation of the part exposed to the user from the functional ROS services part. Unfortunately, NiceGUI uses websockets as a means to provide communication from the frontend to the backend. Hosting two websocket servers would require considerable network reconfiguration, which would make the project even more complicated.

A web application has been chosen as the main way for user interaction as it does not require any sort of installation and allows the user to control the inspection flow from a comfortable office potentially. A web application is generally composed of two parts: frontend and backend, and the development process is complex for modern applications. The NiceGUI framework has been chosen as a way to simplify development as it allows for writing the frontend and backend using



Inspection zone editor

Inspection zone editor is used to define inspection zones, where selected module (inspection handler) should be applied. Each module may have certain rules to define valid inspection zone (how zone should be selected on template, what should be in the zone, etc). Refer module documentation for more details.

Template: Test template reduced

Fig. 8: Open AOI portal, inspection zone editor.

the same language and basically at the same time. The framework provides a set of essential components like buttons, text fields, image elements, and more as Python objects, and when an HTTP request is processed, these objects are translated to a dynamic web application.

The portal is implemented in the style of the MVC model. The portal package defines views, controllers are defined at the core and are used to encapsulate complex logic for the creation of such entities as modules, for example (requiring the creation of database records and uploading blobs to the blob server). Models are defined in the core as well, and they define the database schema.

Short User Manual

The GUI is divided into several main views. The first view that the user faces is the access view, which is designed to authenticate the user against the database by username and password. The next view is a home screen, which displays basic statistics and the last N inspection records. On the left side, the user has the main menu, and at the top of the page, the header is located with the system's general status (indicating the system's availability to perform inspection). The standard flow to set up inspection after login is as follows:

1. Create a camera device. The devices view is used for these purposes. A camera has a title and a description for better identification. Multiple cameras are allowed. The camera may be bound to the GPIO pin so when inspection is triggered from this pin, the related camera is utilized. It is possible to test the camera from this view.
2. Create a template. After the camera is defined, the user will generally want to create a golden image for the inspection. To do so, the user will use the template view and will take an image with one of the defined cameras. The template also has a title and description and may be used with a different camera afterwards.

3. Next, it is important to upload a inspection handler module (if required). The user will go to the modules view and create the required defect type (if their module tests for non-default defects) and then upload the module code as a single Python file with a predefined structure. As a potential improvement of the module system, a module editor may be considered. Modules have titles and descriptions and are related to the inspection zone, designed to detect the related defect type. A module is created as a database record, and the code is simultaneously uploaded to blob storage (the reference to the blob is back-populated to the database).
4. In order to define inspection zones and assign inspection handlers to them (inspection handler is another name for module), the user will go to the template view and open the required template with the inspection zone editor. The inspection zone editor will provide a GUI to navigate over the template image and define inspection zones as rectangles. These inspection zones are related to the template and may not be transferred. The inspection zones are stored in the database with their titles and descriptions and related inspection handlers. The number of inspection zones is not limited.
5. After the inspection zones are defined, it is required to define an inspection profile to relate camera to the product and to provide the so-called environment to the inspection handler. This is what the inspection profile view is for. The mentioned environment is a mechanism to populate the inspection handler with parameters. The environment is basically a multiline text stored in the database, each row of which contains key-value pairs separated with an equality sign. Each inspection handler defines the environment on its own. If the environment is not valid, the inspection will fail. The environment may be changed. The process of identifying the inspection profile, which should be used, is simple: the required profile should be active and will be looked up with the barcode value from the test image when the inspection is triggered.

The final stage of this process is the testing phase or live inspection. The OpenAOI provides capabilities of manual inspection control (instead of automatic triggering by GPIO pin). To do so, the user will go to the inspection view and select the desired camera which they want to use for inspection. After capture, the related profile will be selected automatically by the barcode value.

9.2.4 GPIO Package

GPIO stands for General Purpose IO. The system uses GPIO pins to communicate with upstream systems. To trigger the inspection, the pin is required to be in the up state for at least 0.5s. The GPIO service will iteratively check the pins' state and will dispatch inspection requests. The OK (accept) and NOK (reject) pins are

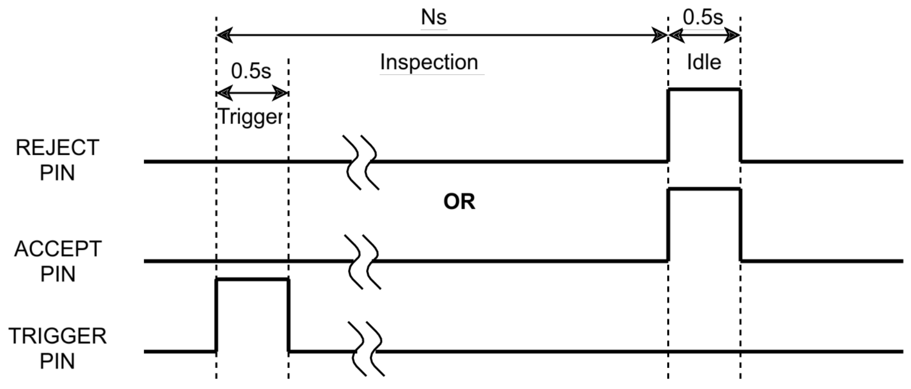


Fig. 9: Open AOI GPIO protocol.

used to report inspection results. The pins will be set to the up state for at least 0.5s. Figure 9 describes the protocol in detail.

9.2.5 Services Package

The services in ROS terminology are small pieces of code that are related to the node, which provides the service, and this code is executed upon request and should return a response as a result. ROS also has a concept of actions, which are service-like entities that, while working on a request, are able to provide feedback to the requester. To implement the core logic of Open AOI, services were chosen as the system should handle particular events (such as new boards to inspect). Should request result in error, this error should be returned in response. In the text, service and nodes are synonymous, however, it is the node that provides the service.

The nodes share the conception of isolating performing dedicated functions, and to bring them together, the mediator pattern was implemented with a mediator node used to interact with interfaces like the web portal or GPIO interface. The following text describes each available service without technical details (like message structure).

Image Acquisition Node

The node is designed to encapsulate camera handling logic under a simple interface. The node provides a simple service to capture an image from a specified camera. Internally, the node will hold the last used camera, and if the next call uses the same camera, no new connection will be made and the image will be captured immediately. Otherwise, the node will connect to the camera first. The node is implemented with the Pylon library.

Product Identification Node

The node is designed to operate on the provided image in order to identify the product. The current version only supports barcode identification. Barcode identification is implemented with the OpenCV barcode detection and decoding library.

Inspection Executor Node

Inspection execution is a node that, when provided with a template image, image under test, a list of inspection zones, Python code as a string to control each zone, and environmental variables, will ensure code interpretation and populate a custom module with all required parameters. The node also ensures correct syntax for the environment string. Logically, this part could be implemented directly inside the mediator node; however, as Open AOI is designed as a distributable system, one may possibly want to execute code with more powerful compute resources, and in this case, inspection execution logic is isolated and minimal changes will be required.

As mentioned earlier, each inspection algorithm is provided as a module. The modules have a very basic interface. Each module is required to define a class that inherit the core **IModule** class to implement its **process** method, which will then receive the tested image and template. A global variable with the name **module** should exist and should contain an instance of the described class. The global variable **DOCUMENTATION** should exist and should contain a string with module information (including environment description). During the inspection the process method will be called and provided with data. It is up to each module how to handle inspection zones, image preprocessing, and so on; the module is required to return a list of inspection log objects of class **IModule.InspectionLog**. Each inspection log should be related to a single inspection zone, and the order should be kept. The inspection log has a **log** property and a **passed** property. The first one should contain an occasional human-readable error description, and the second one is used to indicate the inspection result in an OK/NOK fashion.

Mediator Node

The mediator node conducts management of all other nodes. Its role is to ensure the general inspection flow, as described in figure 10. When an inspection is triggered, the request will contain either a camera ID (when the inspection was triggered manually) or a pin ID (when the inspection was triggered from the GPIO interface). The mediator will then locate the camera record in the database and request image acquisition with this camera. When the test image is captured, the mediator proceeds to identify the board by barcode and will perform a database lookup to retrieve the related inspection profile, which will lead to the template and a list of related inspection zones. After that, the mediator will match inspection zones to inspection handlers and will send each batch of inspection zones to execution.

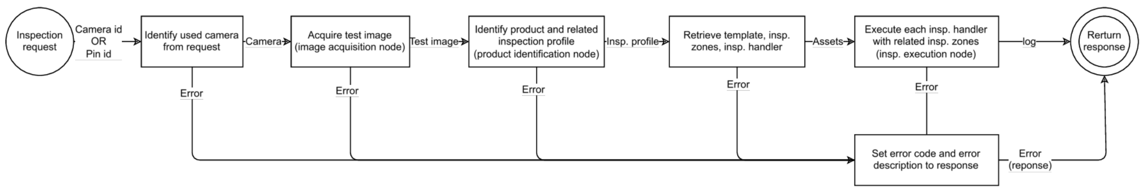

Inspection request
Camera id OR Pin id
Identify used camera from request
Camera
Acquire test image (image acquisition node)
Test image
Identify product and related inspection profile (product identification node)
Insp. profile
Retrieve template, insp. zones, insp. handler
Assets
Execute each insp. handler with related insp. zones (insp. execution node)
log
Rerturn response
Error
Error
Error
Error
Error
Set error code and error description to response
Error (reponse)

Fig. 10: Mediator node main inspection loop.

The response from the execution node is accumulated to be returned at once as the mediator response. Should any error occur during the process, the mediator will terminate and return an error with a description if available.

9.3 Deployment and Development

There are two main ways to deploy Open AOI. The first is to clone the git repository to the target machine with a connected camera and run Open AOI with Docker using the standard Docker Compose command-line interface to bring the system up (see project GitHub for details). The second way is to install ROS2 directly on the target machine, clone the Open AOI repository, compile, and run ROS nodes. To simplify deployment, simple bash scripts were developed. The project build is the first stage and it may be accomplished with **aoi.build.bash**. This script will use the ROS2 CLI to compile ROS packages. The second step is to install Python dependencies. This step may require updating the **aoi.install.bash** script to provide correct paths to ROS folders. The script will call the pip package manager to install required Python dependencies. It is important to notice that ROS2 uses a global Python interpreter and dependencies should be installed after ROS as a site package of one of the Open AOI packages (currently into the core package). The ROS will then hook these packages on startup and they will be available during code interpretation. The last available script that is used to start up the system is **aoi.launch.bash**. If the system is running out of Docker, required dependencies like MySQL and Minio should already be running. The script will source the ROS environment and call the project launch file.

Development of Open AOI requires installing ROS2 on the developer's machine. In order to do so, the developer should follow instructions on the official ROS2 website. Instructions to install ROS2 Foxy distributions can be found at ros.org. Docker installation is also required, and instructions could be found on the Docker official website (docker, docker compose). After the required systems are installed, the process of development is simple and resembles regular deployment. The same build, install, and launch script could be utilized.

10 Experimental Setup

This section describes the experimental setup that was used to develop inspection algorithms. The setup was established for development purposes only, in order to collect datasets. Production setup may vary from the setup described in this section. The experimental setup also utilizes dummy boards for inspection algorithm development, which may differ from those used in production. Collected data is provided as part of this work and is available in the project OSF repository.

10.1 Dummy Boards

In total, 3 dummy boards were used: Network card (further referenced as **netcard**), Drawcore driver board (further referenced as **drawcore**), and Arduino UNO board (further referenced as **arduino**). The boards are not new and some cosmetic issues are present on their surfaces.

10.2 Hardware Configuration

As described in earlier sections, the Open AOI system is designed as a general framework for optical inspection systems. The following setup is the bare minimum example of hardware periphery that may be used with the proposed system.

Core components of the setup are as follows:

- **Camera**. Industrial Basler cameras were used. For monochromatic images, it was the Basler acA2500-60um (5Mp, sensor 12.44mm × 9.83mm, 2590px × 2048px, up to 60fps), and for colorful images, it was the Basler acA2500-60uc (5Mp, sensor 12.44mm × 9.83mm, 2590px × 2048px, up to 60fps). For the experimental setup, a USB interface was used (not supported by the current version of Open AOI).
- **Objective**. A range of objectives were used to determine the best one for inspection needs. For bigger boards, like netcard, a 16mm RICOH objective was used (model FL-BC1618-9M), and for smaller boards like drawcore, a 25mm RICOH was used (FL-BC1618-9M).
- **Illumination**. To avoid shadows, a dome light from Smart View (white) was used (DL-210W), powered by the Smart View multifunctional power source.

In order to determine the size of the smallest detectable feature for a particular objective, the field of view was measured along with the camera working distance, as depicted in figure 12. Technically, this should be done in reverse, but no requirements for the smallest detectable features were defined in advance. The

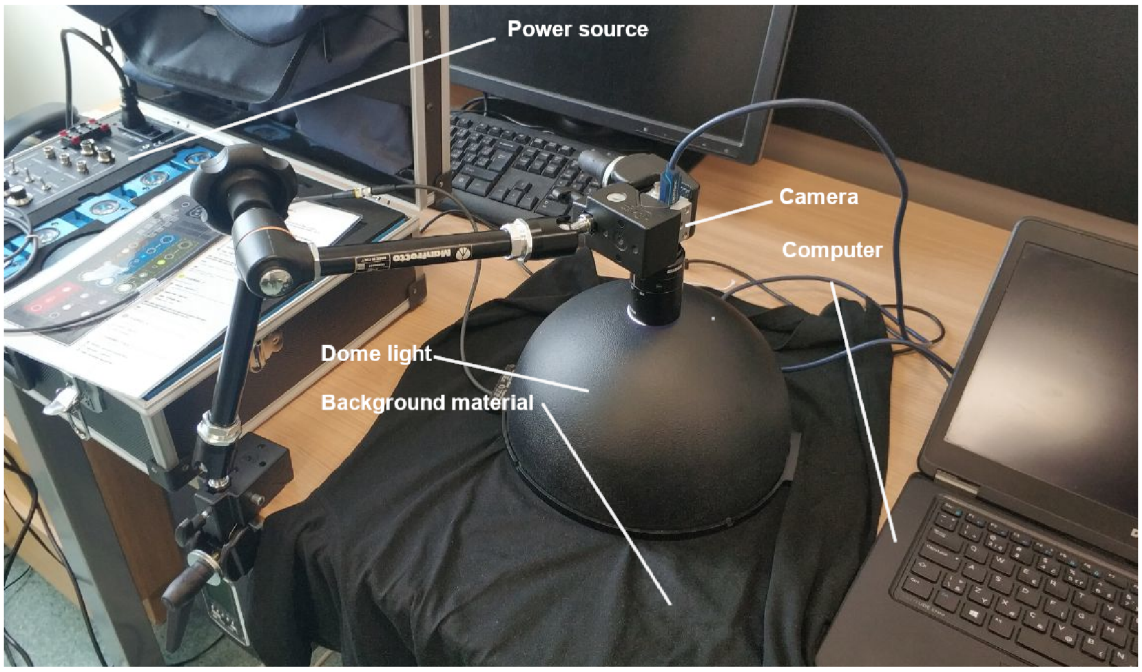


Fig. 11: Experimental setup.

calculated size of the smallest feature will determine if the objective is suitable for certain algorithms, like text recognition, where thin lines should be distinguished. The size of the smallest feature is given by the following equation:

$$O_w = 2\frac{FOV_w}{S_w}, \tag{6}$$

where O_w is the object's smallest feature width in mm, FOV_w is the width of the field of view in mm, and S_w is the sensor width resolution in pixels. The 2 in the formula is related to the Nyquist frequency. Calculated results are presented in table 4.

The calculated feature sizes suggest that the 16mm objective is suitable for missing component detection, as the smallest components on the boards are a few times bigger than the lower bound. However, it may not be suitable for small text recognition tasks, as the line width of some texts, like resistor marks, is less than or close to the limit. With that, text recognition with the 16mm objective is limited to bigger fonts. The same is applicable for the combination of the color camera with the 25mm objective. Due to the Bayer filter, the sensor size is reduced (1295 colorful pixels instead of 2590 monochromatic), and this combination does not provide sufficient resolution for small feature recognition.

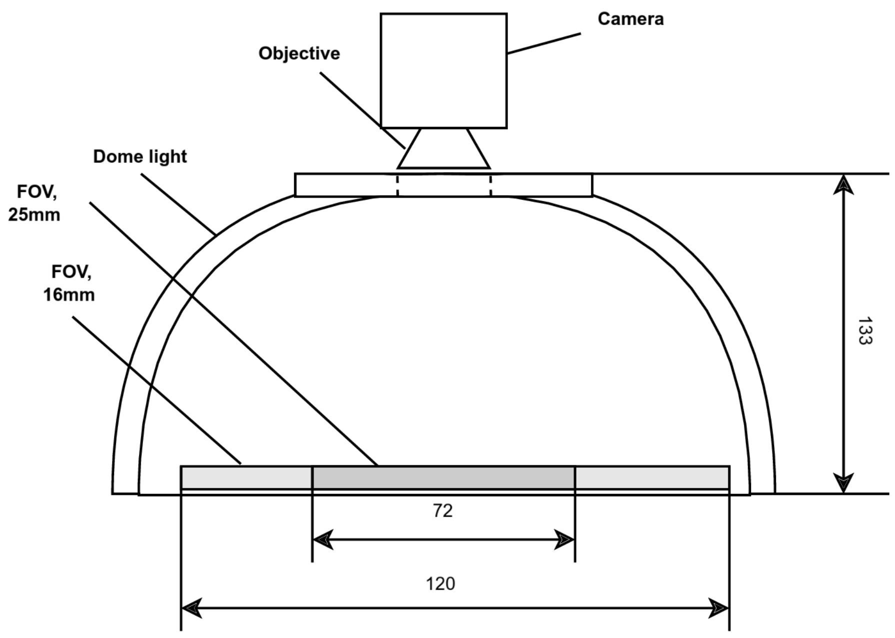


Fig. 12: Experimental setup schema with selected dimensions for resolution calculation.

Objective and sensor	**Working distance**	**FOV**	**Smallest feature size**
16mm, monochromatic sensor	133mm	120mm	0.1mm
25mm, sensor with Bayer filter	133mm	72mm	0.11mm

Tab. 4: Summary of smallest detectable features for experimental setup.

10.3 Collected Dataset

For inspection algorithm development, a sample dataset was collected (download). Only selected combinations of boards with camera setup were processed. For each series and each objective, an F8 aperture was set. Exposure time was selected to simulate different illumination intensity in order to subsequently select the best suitable during algorithm implementation. Exposure values were manually selected to create images with the most balanced illumination, under-illuminated, and over-illuminated images. Boards were slightly shifted manually in random directions to simulate a conveyor belt-based transport system without positioning for inspection needs (which may be the case for smaller manufacturers). The collected dataset is publicly available, and the structure is described by table 5.

Series folder	**Board**	**Camera**	**Objective**	**Exposure time** $[\mu s]$	**Qty**
netcard_um_16mm_f8_4320	Netcard	Mono	16mm	4320	7
netcard_um_16mm_f8_3101	Netcard	Mono	16mm	3101	12
netcard_um_16mm_f8_5014	Netcard	Mono	16mm	5014	10
drawcore_um_16mm_f8_4580	Drawcore	Mono	16mm	4580	12
drawcore_um_16mm_f8_3110	Drawcore	Mono	16mm	3110	12
drawcore_um_16mm_f8_5236	Drawcore	Mono	16mm	5236	12
arduino_um_16mm_f8_5299	Arduino	Mono	16mm	5299	17
arduino_um_16mm_f8_4750	Arduino	Mono	16mm	4750	15
arduino_um_16mm_f8_6023	Arduino	Mono	16mm	6023	13
arduino_uc_16mm_f8_8501	Arduino	Color	16mm	8501	23
arduino_uc_16mm_f8_6138	Arduino	Color	16mm	6138	21
arduino_uc_16mm_f8_10124	Arduino	Color	16mm	10124	22
drawcore_uc_16mm_f8_6448	Drawcore	Color	16mm	6448	25
drawcore_uc_16mm_f8_5270	Drawcore	Color	16mm	5270	29
drawcore_uc_16mm_f8_7604	Drawcore	Color	16mm	7604	36
drawcore_uc_25mm_f8_6009	Drawcore	Color	25mm	6009	34
drawcore_uc_25mm_f8_7473	Drawcore	Color	25mm	7473	27

Tab. 5: Collected dataset structure.

11 Inspection Algorithms

This section provides a close look at the implemented inspections, including performance evaluation of the algorithms.

As described earlier in Chapter 5, the inspection algorithm is the most important part of any AOI. Open AOI's default distribution includes several algorithms that cover missing component defects, print quality defects, and text recognition tasks. From an end-consumer standpoint, each algorithm is a module (inspection handler) that is applied to the whole image and may be applied to each individual inspection zone.

Algorithms for PCB inspection are implemented with the **OpenCV** library for Python [65], while text recognition is handled with the **DocTR** library [66]. OpenCV is an open-source library that provides implementations of basic machine vision algorithms and tools used for image processing. OpenCV has been around for 23 years now (first released in the year 2000 [67]) and is distributed under the Apache License, which makes this open-source project a perfect solution for business applications. DocTR is a collection of OCR neural network models with a simple API around. For text recognition, DocTR adopted the SAR model [68], CRNN model [69], MASTER [70], and other models. Other support libraries may also be used; see the project git repository for details.

11.1 Support Algorithms

11.1.1 Image Alignment

ORB (Oriented FAST and Rotated BRIEF) is an open-source alternative to the popular feature detection algorithm SURF (Speed Up Robust Features). ORB is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance performance. This algorithm is applied whenever alignment of a test image with a template is required. OpenCV implementation is used without modifications. [71]

11.1.2 Board Identification

For the purposes of board identification, an OpenCV implementation of barcode detection and decoding algorithm has been utilized without any modification. [72]

11.2 Missing Component Detection

To cover missing component defects, two approaches were implemented. The first one adopts Discrete Wavelet Transformation to detect missing components in PCB assembly [73] [74] [75]. The second approach utilizes histogram back projection technique to distinguish between component and PCB solder masked base. A similar approach (background subtraction) was taken in [28]. Another alternative approach was considered for implementation - normalized cross-correlation; however, early implementation attempts were not successful.

11.2.1 Discrete Wavelet Transformation

DWT, Discrete Wavelet Transformation, is an image transformation technique for compression (used in JPEG2000) and filtering, often used in machine vision. DWT compression has an important property - it preserves image information in the spatial domain, allowing template matching. Another important feature of this compression technique is high compression ratio and small computational time. The baseline for template matching with DWT is to apply DWT compression and compare compressed test image to compressed template image. Haar wavelet transformation [76] was used [73].

General form of wavelet transformation as a function of a 2D image is the following [76]:

$$W^s(k,l) = \frac{1}{s}\sum_n\sum_m f(m,n)\psi\Big(\frac{m-k}{s}\Big)\psi\Big(\frac{n-l}{s}\Big), \tag{7}$$

where *(k,l)* is the position vector of the wavelet and s is the scale. For Haar wavelet transformation, the wavelet function is defined as follows: [73]

$$\psi(t) = \begin{cases} 1 & \text{if } 0 \le t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \le t < 1 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

For purposes of template matching, Haar transformation is defined as the process of averaging and taking the difference in horizontal and vertical directions. Consider $A = (a_{ij})^{M\times N}$ to be an input image $M \times N$ pixels, where $a_{ij}(1 \le i \le M, 1 \le j \le N)$ is the intensity value of a particular pixel. Then average and difference in the horizontal direction are defined as [73]:

$$s^h_{ij} = \frac{a_{2i-1,j} + a_{2i,j}}{2}, \quad i = 1, ..., \frac{M}{2}, \quad j = 1, ..., N \tag{9}$$

$$d^h_{ij} = \frac{a_{2i-1,j} - a_{2i,j}}{2}, \quad i = 1, ..., \frac{M}{2}, \quad j = 1, ..., N \tag{10}$$

s_{ij}^h and d_{ij}^h are the average (s for *sum*) and difference (d for difference) in the horizontal direction. When similar calculations are applied for the vertical direction, the new image may be constructed from the 4 resulting matrices $\left(\frac{M}{2} \times \frac{N}{2}\right)$ as [73]:

$$B = \begin{pmatrix} B^{UL} & B^{UR} \\ B^{LL} & B^{LR} \end{pmatrix} \tag{11}$$

The B^{UL} image is a compressed version of the original image. Applying the algorithm again to the compressed image will increase the compression ratio. Figure 13 shows the DWT procedure for a sample PCB image.

After the DWT transformation has been applied to both images (test image and template image), the difference between them may be used to detect missing components. Figure 14 shows the result of the difference operation. A median average filter should be applied to the resulting image to reduce noise, and finally, the region of interest should be tested with a threshold value to obtain boolean-type information about missing components. The whole process is described by algorithm 1.

Algorithm 1 DWT Inspection Algorithm

```
Require: $I_{template}$, $I_{test}$, $c$, $DWT\_ITERATIONS$, $THRESHOLD$, $ACCEPTABLE\_DIFFERENCE_{\%}$

Ensure: $pass \in \{true, false\}$
  $c \leftarrow [c_{top}, c_{left}, c_{width}, c_{height}]$    ▷ Inspection zone coordinates
  $I_{template\_cropped} \leftarrow crop(I_{template}, c)$
  $I_{test\_cropped} \leftarrow crop(I_{test}, c)$
  $I_{test\_aligned} \leftarrow align(I_{test\_cropped}, I_{template\_cropped})$ ▷ Align test image with template
  for $i \leftarrow 1, DWT\_ITERATIONS$ do
      $I_{test\_dwt} \leftarrow DWT(I_{test\_aligned})$    ▷ Apply DWT to template
      $I_{template\_dwt} \leftarrow DWT(I_{template\_cropped})$    ▷ Apply DWT to test
      $I_{test\_aligned} \leftarrow I_{test\_dwt}[0 : height/2^i, 0 : width/2^i]$ ▷ Select left upper quadrant
      $I_{template\_cropped} \leftarrow I_{template\_dwt}[0 : height/2^i, 0 : width/2^i]$
  end for
  $I_{diff} \leftarrow threshold(abs(I_{test\_aligned} - I_{template\_cropped}), THRESHOLD)$    ▷ Threshold difference
  $\delta_{\%} \leftarrow \frac{count\_nonzero(I_{diff})}{total\_pixels(I_{diff})} \times 100$    ▷ Calculate percentage difference
  $pass \leftarrow \delta_{\%} < ACCEPTABLE\_DIFFERENCE_{\%}$    ▷ Check if difference is acceptable
  return $pass$
```

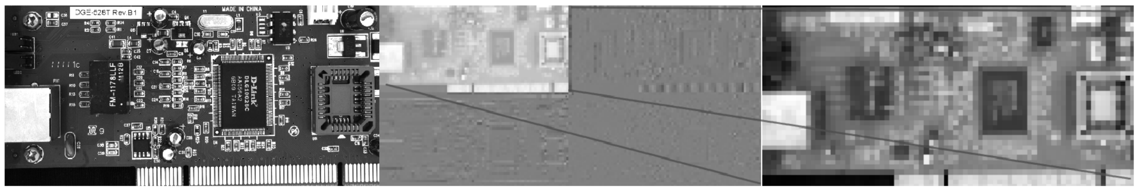


Fig. 13: DWT transformation applied to a PCB image (*network card*).

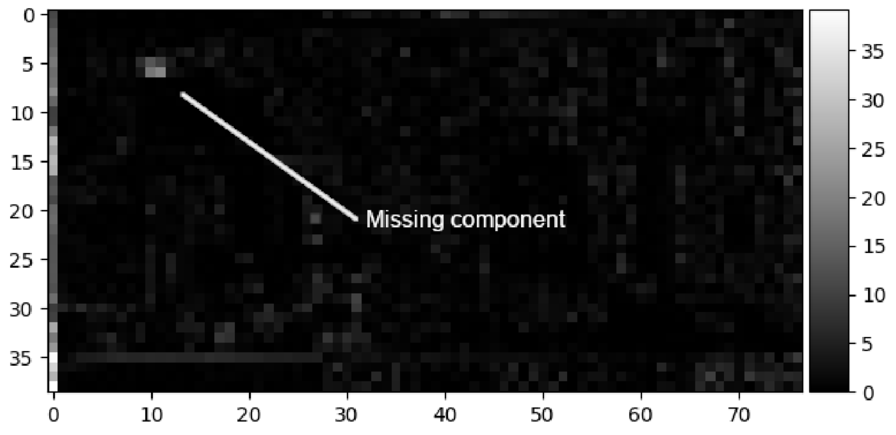


Fig. 14: Difference between the tested PCB image and the template image after DWT compression (*network card*).

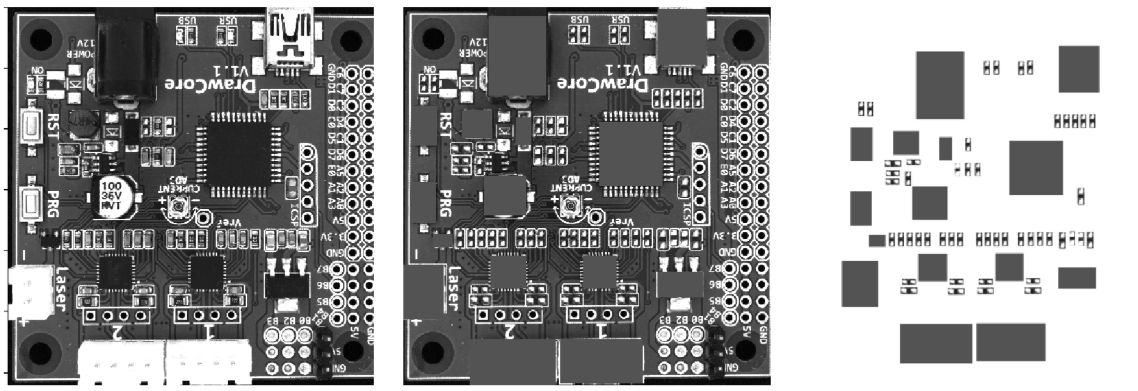


Fig. 15: Synthetic data preparation for DWT algorithm performance assertion (*drawcore*).

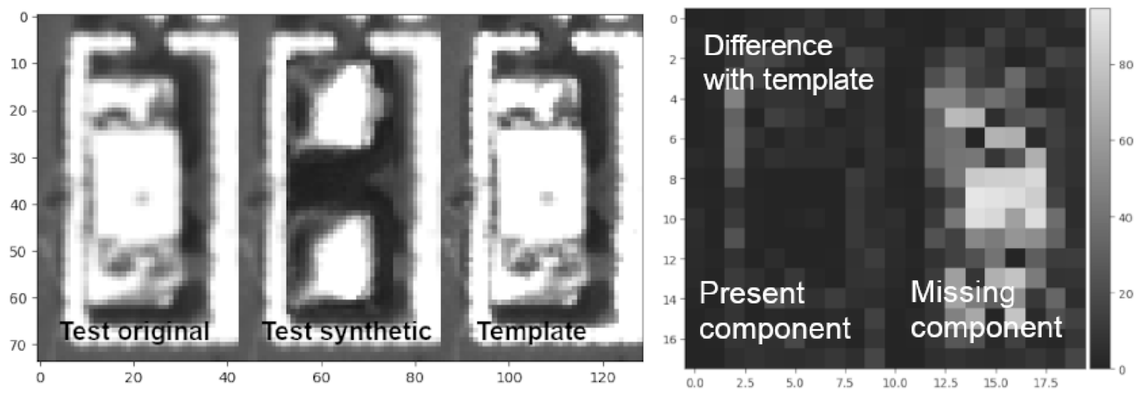


Fig. 16: Example of synthetic data for single inspection zone with DWT transformation applied (*drawcore*).

Dataset	**Images**	**Inspection zones**	**Parameters**	**Accuracy**
Drawcore gray, 16mm, f8, 4580us	12	$2 \times 64/image$	2 iterations, 30 threshold, 10% difference	97%
Drawcore gray, 16mm, f8, 3110us	12	$2 \times 64/image$	2 iterations, 30 threshold, 10% difference	55%

Tab. 6: Summary of DWT performance evaluation and illumination influence impact for constant algorithm parameters.

The method performance (accuracy) has been evaluated with synthetic data obtained by masking components with bare areas of PCB where components are missing. The goal was to keep synthetic data as close to real as possible in terms of statistical features of defects to prevent any impact on test results. To create synthetic data, a template image was chosen, and a mask that hides all components in the best possible way was manually created in GIMP (Figure 15). Algorithm 1, which describes the logic of image inspection with DWT compression, was applied to all available images of Drawcore PCB against the template image. Table 7 summarizes algorithm evaluation results under different light conditions (positive is used for present component, negative for missing). As it may be noticed, the approach is sensitive to illumination conditions - a dramatic reduction of detection power is observed when exposure time changes with unchanged algorithm configuration and template image.

11.2.2 Histogram Backprojection

While exploring existing implementations of AOI, histogram backprojection was not noticed to be used for PCB inspection tasks. However, during experiments with this method, high contrast has been achieved between PCB body and components (including component pads). Histogram backprojection is used for image segmentation to find objects of interest. The main idea behind the algorithm is to compare the histogram of a small area in the image with the known histogram of the object of interest and thus to identify the probability with which each pixel in the image belongs to the object of interest [77].

For the purposes of PCB inspection, the object of interest is defined as the PCB resin body (component background). Under the condition that the PCB body is not of the same color as the components and it covers a greater part of the image than components, a high contrast image may be constructed. Background is then detected by comparing the histogram of the full image with a sliding window. The method was improved by averaging resulting images from different window sizes. The final step is to define the area where the background is allowed (no component expected) and where it is forbidden (a component expected). Algorithm 2 defines the whole process of inspection. The benefit of this algorithm is that it technically does not require a template image as the test image may be used to identify the background. This approach is also fairly intuitive in terms of parameters configuration.

Having a test image I_{test} and a template image $I_{template}$ containing the object we're looking for, calculate a joint histogram for the template image $I_{template}$:

$$H_{template}(b, g, r) = n(b, g, r)/|I_{template}| \tag{12}$$

where b, g, r represent intensity values for a particular bin in the histogram (specific combination of blue, green, and red). $n(b, g, r)$ is the number of pixels in $I_{template}$ with those intensity values, and $|I_{template}|$ is the total number of pixels in the template image.

Then, to perform the backprojection, go through each pixel (i, j) in the test image I_{test}. For each pixel intensity values (b_{ij}, g_{ij}, r_{ij}), calculate a backprojected value $B(i, j)$ using the reference histogram (or PDF):

$$B(i, j) = P_{template}(b_{ij}, g_{ij}, r_{ij}) \tag{13}$$

To evaluate the performance of the inspection process, an accuracy metric has been selected, and the inspection process was tested on real data only. Due to the fact that real PCBs do not have many missing components, the area where the background is observed surrounded by non-background objects was used to simulate missing components. With correct parameter configuration, a 100% accuracy was achieved. Illumination conditions did not have an impact on the result.

Algorithm 2 Histogram Backprojection Inspection Algorithm

```
Require: $I_{template}$, $I_{test}$, $c$, $BINS_{LB}$, $BINS_{UB}$,
    $BACKGROUND\_PROBABILITY\_THRESHOLD$, $KERNEL\_SIZE$,
    $ACCEPTABLE\_BACKGROUND\_RATIO_{\%}$

Ensure: $pass \in \{true, false\}$
  $c \leftarrow [c_{top}, c_{left}, c_{width}, c_{height}]$                ▷ Inspection zone coordinates
  $I_{template} \leftarrow read\_image()$
  $I_{test} \leftarrow read\_image()$
  $I_{test} \leftarrow align(I_{test}, I_{template})$                ▷ Align test image with template
  $HBP_{test} \leftarrow compute\_hbp(I_{test}, BINS_{LB})$
  for $bins \leftarrow BINS_{LB} + 1$ to $BINS_{UB}$ do                ▷ Weighting HBP
      $HBP_{test} \leftarrow HBP_{test} + \frac{compute\_hbp(I_{test}, bins)}{bins}$
  end for
  $HBP_{test} \leftarrow normalize(HBP_{test})$
  $HBP_{test} \leftarrow cut(HBP_{test}, c)$                ▷ Crop HBP image
  for $i \leftarrow 0$ to $n$ do
      for $j \leftarrow 0$ to $m$ do
          if $HBP_{test}[i, j] > BACKGROUND\_PROBABILITY\_THRESHOLD$
  then
              $HBP_{test}[i, j] \leftarrow 1$
          else
              $HBP_{test}[i, j] \leftarrow 0$
          end if
      end for
  end for
  $I_{test} \leftarrow median\_blur(HBP_{test}, KERNEL\_SIZE)$                ▷ Noise reduction
  $background\_ratio_{\%} \leftarrow \frac{sum(I_{test})}{|I_{test}|}$                ▷ Calculate background ratio
  $pass \leftarrow background\_ratio_{\%} < ACCEPTABLE\_BACKGROUND\_RATIO_{\%}$
  ▷ Check if background ratio is acceptable
  return $pass$
```

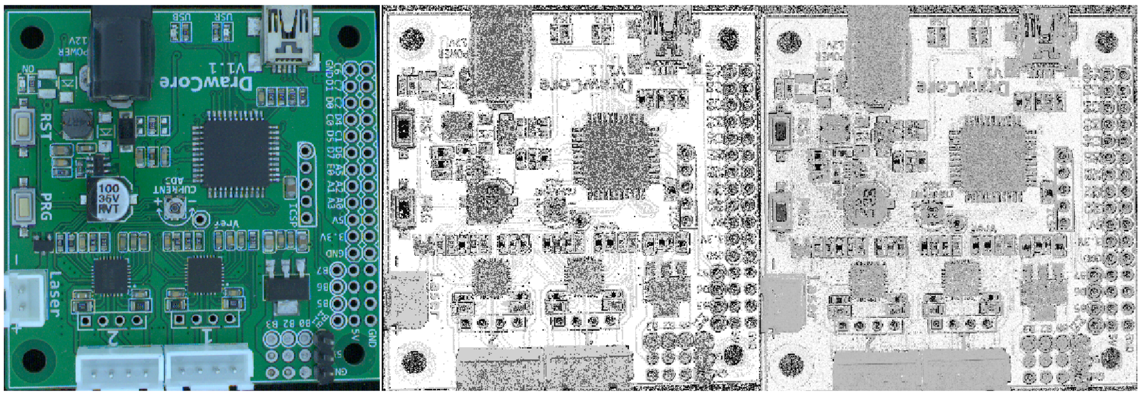


Fig. 17: Example of histogram backprojection applied to a PCB. Left - original image, middle - histogram backprojection with 10 bins histogram, right - weighted histogram backprojection (5 to 45 bins).

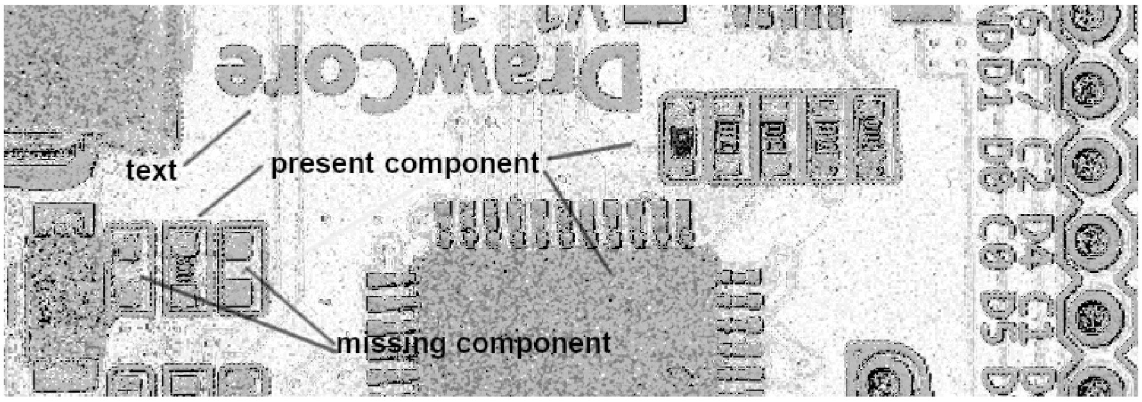


Fig. 18: A closer look at histogram backprojection applied to a PCB.

Dataset	**Images**	**Inspection zones**	**Parameters**	**Accuracy**
Drawcore color, 16mm, f8, exposure: mixed 6448us, 5270us, 7604us	90	$2 \times 17/image$	5 bins LB, 35 bins UB, 3 kernel size, background probability 0.85, acceptable ratio 0.85	100%

Tab. 7: Summary of histogram backprojection performance evaluation.

11.3 Print Quality

Print quality inspection is designed to catch printing issues, like defective characters, missing print, paint-contaminated surfaces, etc. Inspection relies on the fact that printed information is required to have high contrast with the background to be readable by humans. To judge how well printed text matches the template, a simple Otsu threshold technique is used followed by the XOR operation.

In its simplest form, the Otsu method returns a single threshold that separates pixels into two classes: foreground and background. This threshold is selected by minimizing intra-class intensity variance [78].

The intra-class variance is defined as the weighted sum of variances of the two classes:

$$\sigma_w^2(t) = w_0(t)\sigma_0^2(t) + w_1(t)\sigma_1^2(t), \tag{14}$$

where w_0 and w_1 are the probabilities of the two classes separated by a threshold t, and σ_0^2 and σ_1^2 are variances of the two classes.

The class probability $w_{\{0,1\}}(t)$ is computed from N histogram bins:

$$w_{\{0,1\}}(t) = \sum_{i=0}^{t-1} p(i) \tag{15}$$

The next step is the XOR operation, applied to the binary template and test image. The logical XOR is defined as in Table 8.

The resulting image is the full difference of the tested image and the template, and it is possible to define the maximum allowed value. However, this metric suffers from misalignment - when the test image is aligned with the template, there are still mismatches on the details level, which will affect print quality inspection. To minimize this effect, a local alignment was implemented based on 2D cross-correlation of two image chunks (translation only, no rotation).

The cross-correlation is a measure of the similarity of two series as a function of the displacement of one relative to the other. The 2D cross-correlation is defined as follows:

$$(f \star g)[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j] g^*[i-m, j-n], \tag{16}$$

To evaluate the performance of the algorithm, a synthetic dataset has been created with the GNU Image Manipulation Program. This way, a large dataset was available, which made it possible to calculate reliable allowed differences between tested and template images. When 95% reliability of the test is required, the minimal allowed difference should be greater than 8.67% (in other words, it was found that 95% of undamaged test cases have a natural difference with the template under 8.67%, and any greater difference will probably indicate a defect).

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Tab. 8: XOR truth table.

11.4 Capacitor Polarity

The capacitor inspection algorithm is an implementation of the method proposed by Jian Fang et al. [79]. The template image is subtracted from the test image. The result is then binarized with a threshold so that the capacitor orientation mark becomes a white blob. In order to remove potential noise, the erosion operation may be additionally performed. The binary image is expected to have two white blobs if the capacitor in the tested image is rotated; otherwise, no blobs are expected at all.

To assess algorithm performance, a reversed approach has been taken, and the best parameters were identified with grid search to achieve maximum performance. Due to the fact that capacitors on the sample board (and sample images consequently) were contaminated with paint, a white circle was placed in the center of each capacitor to cover the paint and allow algorithm development (clean capacitors are expected to look the same as they have a shiny surface on the top and will reflect light forming the same white circle in the center). To detect the best parameters, images with different exposures were used. The search identified that it is possible to set up the inspection algorithm to achieve 98.27% accuracy. Table 9 provides detailed performance information.

Dataset	**Images**	**Inspection zones**	**Parameters**	**Accuracy (max)**
Netcard gray, 16mm, f8, exposure: mixed 4320us, 3101us, 5014us	29	$2 \times 2/image$	34 threshold, 4 kernel size, 4 erosion iterations	98.27%

Tab. 9: Summary of capacitor orientation performance evaluation.

Algorithm 3 Capacitor Orientation Inspection Algorithm

```
Require: $I_{template}$, $I_{test}$, $c$, $THRESHOLD$, $EROSION\_ITERATIONS$,
    $KERNEL\_SIZE$

Ensure: $pass \in \{true, false\}$
    $c \leftarrow [c_{\text{top}}, c_{\text{left}}, c_{\text{width}}, c_{\text{height}}]$    ▷ Inspection zone coordinates
    $I_{template} \leftarrow read\_image()$
    $I_{test} \leftarrow read\_image()$
    $I_{test} \leftarrow align(I_{test}, I_{template})$    ▷ Align test image with template
    $I_{test} \leftarrow cut(I_{test}, c)$
    $I_{template} \leftarrow cut(I_{template}, c)$
    $I_{diff} \leftarrow absdiff(I_{test}, I_{template})$
    $I_{diff} \leftarrow threshold(I_{diff}, THRESHOLD)$    ▷ Threshold difference
    for $i \leftarrow 1$ to $EROSION\_ITERATIONS$ do
        $I_{diff} \leftarrow erode(I_{diff}, KERNEL\_SIZE)$    ▷ Erode the difference image
    end for
    $number\_of\_connected\_components \leftarrow count\_connected\_components(I_{diff})$    ▷
    Count connected components
    if $number\_of\_connected\_components < 2$ then
        $pass \leftarrow true$
    else
        $pass \leftarrow false$
    end if
    return $pass$
```

11.5 Text Recognition

The text recognition task is considered for board identification, component identification, etc. This feature is not currently fully supported by the framework and was developed mainly for experimental purposes. OCR for industrial purposes does not differ much from conventional OCR for any other purpose, like document digitalization. With that, Open AOI utilizes the open-source library DocTR for Python, which offers character recognition. The library utilizes neural networks wrapped into a simple, yet efficient Python interface. The library was tested for its ability to read text on the board correctly. The result of the test was that around 83% of texts were identified correctly. This result is not enough for the production usage of text recognition with DocTR, as many components and texts will not be identified correctly, resulting in false alarms.

12 System Performance

Section is dedicated to performance assertion of the Open AOI system as a whole and to different aspects of its practical usage. The currently targeted deployment platform is Raspberry Pi. Test deployment has been performed on Raspberry Pi 5 and results will be discussed.

System performance is described by the time cost of a single inspection. The system's performance is directly related to the power of the hosting compute. To understand the characteristics of the Open AOI framework in terms of performance, simple profiling was conducted. Profiling is a technique that allows obtaining the duration (or level of consumption of any other resource, like RAM) of the process or function.

Initial profiling was conducted for the system in emulator mode (camera hardware is emulated) on an Intel Core i7-5600U CPU @ 2.60GHz × 4 (personal laptop), and then profiling on the targeted Raspberry Pi 5 was also performed (4-bit 2.4 GHz quad-core ARM Cortex-A76). The template image and test image have sizes of 1200px × 1200px (in this case, the Drawcore image was used, and by manually cropping the image for the test, a production camera setup was simulated). For profiling, mock inspection algorithms were used (the algorithm accepted all inspection zones without any tests). The goal of profiling was to identify the system's general bottlenecks (not related to hardware or inspection algorithms). The result is described in Figures 19 and 20. As it may be observed, the total processing time of the system on a personal computer is around 2.5-3s, which satisfies the requirements. On the target Raspberry platform, the time totals 5-5.5s, which is slightly above the requirements. The biggest time losses are related to image manipulations and ROS2-related internals. It was noticed that the image size is, in fact, significant for the system performance. In order to improve system performance further, it is recommended to reduce the region of interest when setting up the camera for the project. Another way of improving performance involves Open AOI source code modification - the fastest processing time would be achieved when image acquisition

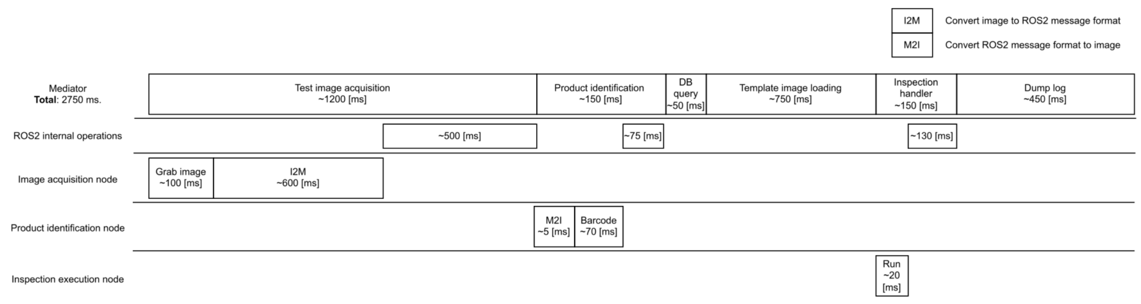

I2M Convert image to ROS2 message format
M2I Convert ROS2 message format to image
Mediator Total: 2750 ms.
Test image acquisition ~1200 [ms]
Product identification ~150 [ms]
DB query ~50 [ms]
Template image loading ~750 [ms]
Inspection handler ~150 [ms]
Dump log ~450 [ms]
ROS2 internal operations
~500 [ms]
~75 [ms]
~130 [ms]
Image acquisition node
Grab image ~100 [ms]
I2M ~600 [ms]
Product identification node
M2I ~5 [ms]
Barcode ~70 [ms]
Inspection execution node
Run ~20 [ms]

Fig. 19: Open AOI profile (emulation mode, personal computer).

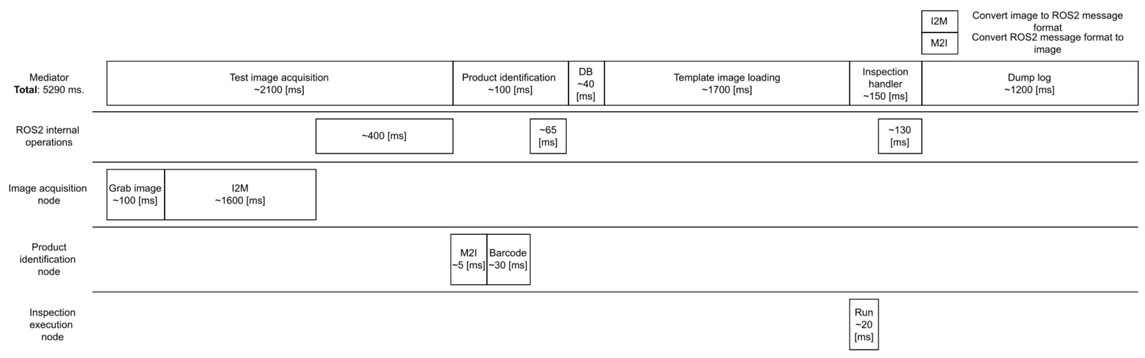


Fig. 20: Open AOI profile (emulation mode, Raspberry Pi 5).

and inspection handler execution are in a single ROS2 node. This, however, will break the concept of Open AOI. Raspberry Pi may also not be the best choice to deploy the system, as the dual-core Raspberry is struggling to serve project needs (isolation of database services and blob storage may improve the situation).

13 Conclusion

The work presented the theoretical basis and practical implementation of an open-source optical inspection system.

Regarding the theoretical aspect of the work, the current state of the optical inspection industry has been elucidated. Resources pertaining to selected inspection issues and the general inspection process were identified and processed to encompass the entire inspection procedure.

The practical segment of the work culminates in the implementation of the Open AOI system, which is now publicly available. This system provides a framework for common optical inspection tasks. Open AOI meets end-consumer-defined requirements in terms of inspection capabilities, performance, and security aspects. The final stage of the practical phase involves deploying the system to the original end-consumer. However, this stage is yet to be completed as final negotiations with the end-consumer are still in progress. As for on-site implementation of Open AOI, a balanced selection of image quality and size of region-of-interest (ROI) has to be considered to address potential inspection rate decrease. One may also want to offload data storage from the inspection unit to free up CPU resources for inspection purposes (as profiling suggests).

Various algorithms were implemented to address specific PCB defect types, such as missing components, capacitor orientation, print quality, etc. It is important to notice that for real application algorithms will require fine-tuning, as in the case of every other machine vision task. Otherwise, experiments have shown that it is possible to achieve reliable (more then 95% accuracy, except for text recognition) results using the proposed implementations. The proposed setup of dome light with Basler industrial camera also proved to be sufficient for the solved task.

Throughout the implementation process, a dataset containing over 300 images was created and is now also available to the public.

The implemented algorithms and their related theories, when considered independently from the Open AOI system, may serve as a solid starting point for anyone interested in optical inspection for PCBs. The implementation of algorithms follows related research papers, and in conjunction with the collected dataset and the developed algorithm performance evaluation environment, it may eventually contribute to a better understanding of the implemented algorithms and the aspects of their applicability and limitations.

14 BIBLIOGRAPHY

[1] SOOD, B. *Printed circuit board inspection and quality control - PCB failure causes and cures.* GSFC-E-DAA-TN59251. Greenbelt, MD, United States: NASA Goddard Space Flight Center, August 2018. [online]. [accessed: 21 February 2024]. Available from: https://ntrs.nasa.gov/citations/20180005658.

[2] MITRA, A. *Fundamentals of Quality Control and Improvement.* 3rd ed. Hoboken, NJ: Wiley, 2008. ISBN 978-1-118-70514-8.

[3] EBAYYEH, A. A. R. M. A. and MOUSAVI, A. A Review and Analysis of Automatic Optical Inspection and Quality Monitoring Methods in Electronics Industry. *IEEE Access.* 2020, vol. 8, p. 183192–183271. DOI: 10.1109/ACCESS.2020.3029127.

[4] IPC. *IPC-T-50N, Terms and Definitions for Interconnecting and Packaging Electronic Circuits.* Bannockburn, 2021.

[5] CALTRONICS DESIGN & ASSEMBLY, INC.. *PCB Inspection and Testing Methods - Caltronics PCB Design & Assembly* [no date]. [online]. Available from: https://www.caltronicsdesign.com/white-papers/pcb-inspection-and-testing-methods [accessed: 14 March 2024].

[6] BRINDLEY, K. *Newnes electronics assembly handbook.* Oxford, England: Butterworth-Heinemann, 1990.

[7] CHIN, R. T. and HARLOW, C. A. Automated Visual Inspection: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 1982, PAMI-4, no. 6, p. 557–573. DOI: 10.1109/TPAMI.1982.4767309.

[8] MOGANTI, M. and ERCAL, F. Automatic PCB inspection systems. *IEEE Potentials.* 1995, vol. 14, no. 3, p. 6–10. DOI: 10.1109/45.464686.

[9] CANTONE, C. and FARO, A. An Overview of the Automated Optical Inspection Edge AI Inference System Solutions. In:. February 2024, p. 153–175. DOI: 10.1201/9781003478713-7. ISBN 9781003478713.

[10] ZEITHAML, V. A., GREMLER, D. D. and BITNER, M. J. *Services Marketing: Integrating Customer Focus Across the Firm.* Hardcoverth ed. McGraw-Hill Education, 2017. ISBN 0078112109,9780078112102. Available at: http://gen.lib.rus.ec/book/index.php?md5=5293a41d95cdac92bcb2a303407f8d4a.

[11] WIKIPEDIA CONTRIBUTORS. *Computer vision* [https://en.wikipedia.org/w/index.php?title=Computer_vision&oldid=1215049085]. Mar 2024. [accessed: 10 April 2024].

[12] BATCHELOR, B. and WALTZ, F. *Intelligent Machine Vision - Techniques, Implementations and Applications*. January 2001. ISBN 978-3-540-76224-9.

[13] LABUDZKI, R., LEGUTKO, S. and RAOS, P. The essence and applications of machine vision. *Tehnicki Vjesnik*. 2014, vol. 21, no. 4, p. 903–909.

[14] TORRAS, C. *Computer Vision: Theory and Industrial Applications*. Springer Science & Business Media, 2012.

[15] BATCHELOR, B. G., ed. *Machine Vision Handbook*. 2012th ed. London, England: Springer, 2012.

[16] MARTIN, D. A practical guide to machine vision lighting. *Midwest Sales and Support Manager, Adv Illum2007*. 2007, p. 1–3.

[17] LITWILLER, D. Ccd vs. cmos. *Photonics spectra*. 2001, vol. 35, no. 1, p. 154–158.

[18] CENZANO, J. *Camera model* [https://jordicenzano.name/front-test/2d-3d-paradigm-overview-2011/camera-model/]. Jun 2014. [accessed: 30 March 2024].

[19] KENJI HATA, S. S. *CS231A: Computer vision, from 3D reconstruction to recognition* [https://web.stanford.edu/class/cs231a/course_notes.html]. [accessed: 30 March 2024].

[20] JAIN, R. C., KASTURI, R. and SCHUNCK, B. G. *Machine Vision*. Maidenhead, England: McGraw Hill Higher Education, 1995.

[21] WHITAKER, J. C., ed. *The electronics handbook, second edition*. 2nd ed. Boca Raton, FL: CRC Press, 2005.

[22] WORLD, M. R. *Global Industrial Machine Vision Camera Market 2024 By Manufacturers, Regions, Type And Application, Forecast To 2030*. Nov 2024.

[23] HOTAŘ, V. *Introduction To Machine Vision Fundamental Principles And Hardware*. Czech Republic, Liberec: Technical University of Liberec, 2017.

[24] MALAMAS, E., PETRAKIS, E., ZERVAKIS, M., PETIT, L. and LEGAT, J.-D. A survey on industrial vision systems, applications and tools. *Image and Vision Computing*. february 2003, vol. 21, p. 171–188. DOI: 10.1016/S0262-8856(02)00152-X.

[25] Iano, Y., Bonello, D. and Neto, U. Text recognition in PCBs: an object character recognition (OCR) algorithm. *International Journal of Development Research*. july 2020, vol. 10, no. 7, p. 1–7. DOI: 10.37118/ijdr.19567.07.2020.

[26] Ling, Q. and Mat Isa, N. A. Printed Circuit Board Defect Detection Methods Based on Image Processing, Machine Learning and Deep Learning: A Survey. *IEEE Access*. january 2023, PP, p. 1–1. DOI: 10.1109/ACCESS.2023.3245093.

[27] Hassanin, A.-A. I. M., El Samie, F. E. A. and El Banby, G. M. A real-time approach for automatic defect detection from PCBs based on SURF features and morphological operations. *Multimedia Tools and Applications*. 2019, vol. 78, p. 34437–34457. Available at: https://api.semanticscholar.org/CorpusID:204330448.

[28] Sundaraj, K. PCB inspection for missing or misaligned components using background subtraction. *WSEAS Transactions on Information Science and Applications archive*. 2009, vol. 6, no. 5, p. 778–787. Available at: https://api.semanticscholar.org/CorpusID:17814362.

[29] Tsai, D.-M., Lin, C.-T. and Chen, J.-F. The evaluation of normalized cross correlations for defect detection. *Pattern Recognition Letters*. november 2003, vol. 24, p. 2525–2535. DOI: 10.1016/S0167-8655(03)00098-9.

[30] Annaby, M. H., Fouda, Y. M. and Rushdi, M. A. Improved Normalized Cross-Correlation for Defect Detection in Printed-Circuit Boards. *IEEE Transactions on Semiconductor Manufacturing*. 2019, vol. 32, no. 2, p. 199–211. DOI: 10.1109/TSM.2019.2911062.

[31] Cho, H.-J. and Park, T.-H. Wavelet transform based image template matching for automatic component inspection. *The International Journal of Advanced Manufacturing Technology*. Springer. 2010, vol. 50, p. 1033–1039.

[32] Belbachir, A., Lera, M., Fanni, A. and Montisci, A. An automatic optical inspection system for the diagnosis of printed circuits based on neural networks. In: *Fourtieth IAS Annual Meeting. Conference Record of the 2005 Industry Applications Conference, 2005.* 2005, vol. 1, p. 680–684 Vol. 1. DOI: 10.1109/IAS.2005.1518381.

[33] Crispin, A. and Rankov, V. Automated inspection of PCB components using a genetic algorithm template-matching approach. *International Journal of Advanced Manufacturing Technology*. december 2007, vol. 35, p. 293–300. DOI: 10.1007/s00170-006-0730-0.

[34] LI, D., LI, C., CHEN, C. and ZHAO, Z. Semantic Segmentation of a Printed Circuit Board for Component Recognition Based on Depth Images. *Sensors*. 2020, vol. 20, no. 18, p. 5318. DOI: 10.3390/s20185318. ISSN 1424-8220. Available at: https://www.mdpi.com/1424-8220/20/18/5318.

[35] JESSURUN, N., DIZON PARADIS, O. P., HARRISON, J., GHOSH, S., TEHRANIPOOR, M. M. et al. FPIC: A Novel Semantic Dataset for Optical PCB Assurance. *ACM Journal on Emerging Technologies in Computing Systems*. Apr 2023, vol. 19, no. 2, p. 1–21. DOI: 10.1145/3588032. ISSN 1550-4840. Available at: http://dx.doi.org/10.1145/3588032.

[36] DING, R., DAI, L., LI, G. and LIU, H. TDD-Net: A Tiny Defect Detection Network for Printed Circuit Boards. *CAAI Transactions on Intelligence Technology*. may 2019, vol. 4. DOI: 10.1049/trit.2019.0019.

[37] HUANG, W., WEI, P., ZHANG, M. and LIU, H. HRIPCB: a challenging dataset for PCB defects detection and classification. *The Journal of Engineering*. may 2020, vol. 2020. DOI: 10.1049/joe.2019.1183.

[38] SHEN, J., LIU, N. and SUN, H. Defect detection of printed circuit board based on lightweight deep convolution network. *IET Image Processing*. 2020, vol. 14, no. 15, p. 3932–3940. DOI: https://doi.org/10.1049/iet-ipr.2020.0841. Available at: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-ipr.2020.0841.

[39] MAKWANA, D., MITTAL, S. et al. PCBSegClassNet—A light-weight network for segmentation and classification of PCB component. *Expert Systems with Applications*. Elsevier. 2023, vol. 225, p. 120029. DOI: 10.1016/j.eswa.2023.120029.

[40] B, A. D. and RAO, M. SMT Component Inspection in PCBA's using Image Processing Techniques. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*. 2019, vol. 8, no. 12. ISSN 2278-3075.

[41] LI, W., NEULLENS, S., BREIER, M., BOSLING, M., PRETZ, T. et al. Text recognition for information retrieval in images of printed circuit boards. In: *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*. 2014, p. 3487–3493. DOI: 10.1109/IECON.2014.7049016.

[42] GANG, S., FABRICE, N., CHUNG, D. and LEE, J. Character Recognition of Components Mounted on Printed Circuit Board Using Deep Learning. *Sensors*. 2021, vol. 21, no. 9, p. 2921. DOI: 10.3390/s21092921. ISSN 1424-8220. Available at: https://www.mdpi.com/1424-8220/21/9/2921.

[43] Dhapte, A. *Automated optical inspection system market research report information by component (software, system), by technology (in-line AOI, offline AOI), by type (2D AOI systems, 3D AOI systems), by end user (automotive, aerospace & defense) and by region (north america, europe, asia-pacific, middle east & africa, and south america) - forecast till 2030* [https://www.marketresearchfuture.com/reports/automated-optical-inspection-system-market-5979]. 2024. [online]. Pune, India: Market Research Future, February 2024 [accessed: 17 February 2024]. MRFR/SEM/4521-HCR.

[44] *Automated Optical Inspection (AOI) Equipment - PCB Directory* [Largest Directory of PCB Manufacturers / Fabricators, Assembly and Design Companies]. No date. [online]. [accessed: 17 February 2024]. Available from: https://www.pcbdirectory.com/equipment/search/automated-optical-inspection-equipment.

[45] *PCB Directory | LinkedIn* [LinkedIn]. No date. [online]. [accessed: 17 February 2024]. Available from: https://www.linkedin.com/company/pcbdirectory.

[46] HARTLEY, R. and ZISSERMAN, A. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge UK: Cambridge University Press, 2003.

[47] *AOI Viscom S3088 Ultra series* [Sincotron]. No date. [online]. [accessed: 17 February 2024]. Available from: https://sincotron.eu/product/aoi-viscom-s3088-ultra-series/.

[48] *Parmi. Xceed* [PARMI]. No date. [online]. [accessed: 17 February 2024]. Available from: https://parmi.com/xceed/.

[49] *SMT-JUKI / Surface Mount Technology System | JUKI AUTOMATION SYSTEMS CORPORATION* [SMT-JUKI / Surface Mount Technology System]. No date. [online]. [accessed: 17 February 2024]. Available from: https://www.juki.co.jp/smt/en/.

[50] Wu, F. and Zhang, X. An inspection and classification method for chip solder joints using color grads and Boolean rules. *Robotics and Computer-Integrated Manufacturing*. october 2014, vol. 30, p. 517–526. DOI: 10.1016/j.rcim.2014.03.003.

[51] OMRON Industrial Automation. *VT-S730 PCB inspection system/features*. No date. [online]. Available from: https://www.ia.omron.com/products/family/3390/ [accessed: 18 February 2024].

[52] MAGIC-RAY TECHNOLOGY. *2D AOI-V5000 | product center* [no date]. [online]. Available from: https://en.magic-ray.com/productinfo3.html [accessed: 18 February 2024].

[53] *ACROTEC INTERNATIONAL Co., Ltd.* [no date]. [online]. Available from: http://www.acro-tec.com.tw/aoi01001_details.html?category_id=35%20&amp;item_id=7 [accessed: 18 February 2024].

[54] MAKER RAY. *Products. Maker-ray* [no date]. [online]. Available from: http://www.maker-ray.com/en/product_1 [accessed: 18 February 2024].

[55] SHENZHEN JAGUAR AUTOMATION EQUIPMENT CO., LTD. *Jaguar Off-line AOI Inspection Machine A1000 -Automatic Optic Inspection-Shenzhen Jaguar Automation Equipment Co., Ltd-reflow oven wave soldering machine manufacturer* [no date]. [online]. Available from: http://www.jaguar-ele.net/te_product_f/2008-08-04/189.chtml [accessed: 18 February 2024].

[56] *AOI systems. AOI | Home* [no date]. [online]. Available from: http://www.aoisystems.com/index.html [accessed: 18 February 2024].

[57] YINGXING. *China SMT online AOI machine YX800-OL manufacturers, suppliers - factory direct price - yingxing. China Reflow Oven, Stencil Printer, BGA Rework Station Manufacturers, Suppliers, Factory - Yingxing* [no date]. [online]. Available from: https://www.zjyingxing.com/offline-aoi-machine/online/smt-offline-aoi-machine-yx800-ol.html [accessed: 18 February 2024].

[58] NEODEN SMT. *NeoDen ND800 Online AOI Machine-NeoDen SMT. SMT machine Manufactory, Reflow oven supplier* [no date]. [online]. Available from: https://www.neodentech.com/productinfo/747443.html [accessed: 18 February 2024].

[59] ZHENHUAXING. *Automatic Optical Inspection (AOI) VCTA-Z5(Off-Line) - ZhenHuaXing* [no date]. [online]. Available from: https://aoi-spi.com/product/automatic-optical-inspection-aoi-vcta-z5off-line/ [accessed: 18 February 2024].

[60] OCIRTECH. *Hurricane H-480i — OCIRTech* [no date]. [online]. Available from: https://www.ocirtech.com/products/hurricane-h-480i [accessed: 18 February 2024].

[61] SHENZHEN J-WIDE ELECTRONICS EQUIPMENT CO.,LTD. *AOI machine with online/offline for option in SMT assembly line* [no date]. [online]. Available from: https://www.jwide-smt.com/products/jw-600-online-aoi-machine/ [accessed: 18 February 2024].

[62] AMS, INC. *Odd form parts AOI - AMS, inc.* [no date]. [online]. Available from: https://www.ams-fa.com/product/odd-form-parts-aoi/ [accessed: 18 February 2024].

[63] *Why ROS?* [https://www.ros.org/blog/why-ros/]. [accessed: 9 April 2024].

[64] MARUYAMA, Y., KATO, S. and AZUMI, T. Exploring the performance of ROS2. In:. October 2016, p. 1–10. DOI: 10.1145/2968478.2968502.

[65] *OpenCV - open Computer Vision library* [https://opencv.org/]. Feb 2021. [accessed: 10 April 2024].

[66] *DocTR documentation* [https://mindee.github.io/doctr/latest/index.html]. [accessed: 10 April 2024].

[67] WIKIPEDIA CONTRIBUTORS. *OpenCV* [https://en.wikipedia.org/w/index.php?title=OpenCV&oldid=1208982530]. Feb 2024.

[68] LI, H., WANG, P., SHEN, C. and ZHANG, G. Show, Attend and Read: A Simple and Strong Baseline for Irregular Text Recognition. *CoRR*. 2018, abs/1811.00751. Available at: http://arxiv.org/abs/1811.00751.

[69] SHI, B., BAI, X. and YAO, C. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. *CoRR*. 2015, abs/1507.05717. Available at: http://arxiv.org/abs/1507.05717.

[70] LU, N., YU, W., QI, X., CHEN, Y., GONG, P. et al. MASTER: Multi-Aspect Non-local Network for Scene Text Recognition. *CoRR*. 2019, abs/1910.02562. Available at: http://arxiv.org/abs/1910.02562.

[71] RUBLEE, E., RABAUD, V., KONOLIGE, K. and BRADSKI, G. ORB: An efficient alternative to SIFT or SURF. In: *2011 International Conference on Computer Vision*. 2011, p. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[72] WANG, J. L. . *Recognizing one-dimensional barcode using* [https://opencv.org/blog/recognizing-one-dimensional-barcode-using-opencv/]. Jun 2021. [accessed: 1 May 2024].

[73] CHO, H.-J. and PARK, T.-H. Wavelet transform based image template matching for automatic component inspection. *The International Journal of Advanced Manufacturing Technology*. october 2010, vol. 50, p. 1033–1039. DOI: 10.1007/s00170-010-2567-9.

[74] IBRAHIM, Z. and AL ATTAS, S. Wavelet-based printed circuit board inspection algorithm. *Integrated Computer-Aided Engineering*. april 2005, vol. 12, p. 201–213. DOI: 10.3233/ICA-2005-12206.

[75] LIU, H., ZHOU, W., KUANG, Q., CAO, L. and GAO, B. Defect detection of IC wafer based on two-dimension wavelet transform. *Microelectron. J.* 2010, vol. 41, p. 171–177. Available at: https://api.semanticscholar.org/CorpusID:2985889.

[76] ZHANG, D. Wavelet Transform. In: *Fundamentals of Image Data Mining: Analysis, Features, Classification and Retrieval*. Cham: Springer International Publishing, 2019, p. 35–44. DOI: 10.1007/978-3-030-17989-2_3. ISBN 978-3-030-17989-2. Available at: https://doi.org/10.1007/978-3-030-17989-2_3.

[77] SWAIN, M. and BALLARD, D. Indexing via color histograms. In: *[1990] Proceedings Third International Conference on Computer Vision*. 1990, p. 390–393. DOI: 10.1109/ICCV.1990.139558.

[78] OTSU, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, vol. 9, no. 1, p. 62–66. DOI: 10.1109/TSMC.1979.4310076.

[79] FANG, J., SHANG, L., GAO, G., XIONG, K. and ZHANG, C. Capacitor Detection on PCB Using AdaBoost Classifier. *Journal of Physics: Conference Series*. IOP Publishing. sep 2020, vol. 1631, no. 1, p. 012185. DOI: 10.1088/1742-6596/1631/1/012185. Available at: https://dx.doi.org/10.1088/1742-6596/1631/1/012185.

LIST OF APPENDICES

A Appendix 1 - Open AOI

A reduced version of Open AOI is attached to this work as `open-aoi-0.1.0.zip`. The provided version is missing images for algorithm evaluation due to space constraints. The full version of Open AOI is also available on GitHub.