Berliner Hochschule für Technik

Master Thesis

Use of Convolutional Neural Networks and Traditional Techniques for Automatic Visual Inspection of Assembled Circuit Boards

Einsatz von Convolutional Neural Networks und Traditionellen Techniken zur automatischen visuellen Prüfung von bestückten Leiterplatten

Aiman Mohamed Ehab Fouad Shahin
935300

Berliner Hochschule für Technik
Fachbereich VI
Technische Informatik-Embedded Systems (M.Eng.)

Examination Commission:
Dr. Eng. Dirk Holm
Prof. Dr. Volker Sommer

Date of Submission: 22 July 2024

Abstract

This thesis explores the integration of traditional image processing techniques and Convolutional Neural Networks (CNNs) for the automatic visual inspection of printed circuit boards (PCBs). The objective is to enhance the accuracy and efficiency of PCB inspections, thus minimising the need for manual verification. The research focuses on leveraging the precise localisation capabilities of traditional methods and the advanced classification abilities of CNNs, particularly the YOLO (You Only Look Once) model. In this Thesis, three inspection methods were created: one that uses the conventional technique, another that uses the CNN model and a third that uses both techniques.

A comprehensive two datasets derived from high-quality PCB images were prepared. One was built by creating artificially generated defects, and the other was prepared by dividing the PCB images into sub-images. The dataset was divided into training and validation sets, and two distinct training methods were employed to optimise the YOLO model. The first training method resulted in a model with a mean Average Precision (mAP) of 99.5%. On the other hand, the second training method, which involved dividing images into smaller sub-images, resulted in a mAP of 99.5%.

The results demonstrated that while CNNs significantly improve component classification, accurate component localisation remains a challenge. Integrating traditional techniques with Artificial Intelligence (AI) models effectively reduced false results and enhanced inspection reliability. However, the combined method slightly increased processing time.

In conclusion, this study highlights the potential of combining traditional and AI-based techniques for PCB inspection. The findings suggest continuous improvement and learning mechanisms, consistent image capture conditions, and clear reference points are essential for achieving high inspection accuracy. The study also found that using the traditional method alone produces unsatisfactory results. Future work should focus on refining these methods and addressing the limitations of AI-based inspection systems, such as the need for larger training datasets and precise coordinate matching.

To my father, in loving memory.

Acknowledgements

First and foremost, I want to thank God for blessing me with this challenge and opportunity to add value. I would like to thank all the people who have been a part of my journey in attaining a master's degree in Germany. This journey has been an experience that has made me grow in all aspects, and I am thankful for all the people who have been a part of it. I also want to express my deepest gratitude to my family for all the unwavering support they have provided me throughout the years. I would also like to thank Helen Dgheim for her incredible support. With her help, I overcame numerous challenges while completing my master's degree. Finally, I extend my heartfelt thanks to Dr. Eng. Dirk Holm for his invaluable guidance throughout the thesis. His feedback and insights were crucial in building this work.

Contents

List of Figures

List of Tables

Abbreviations

2D	Two Dimensional
3D	Three Dimensional
AI	Artificial Intelligence
AOI	Automated Optical Inspection
API	Application Programming Interface
AXI	Automated X-ray Inspection
BF	Bilateral Filter
BFMatcher	Brute Force Matcher
BGA	ball grid arrays
BGR	Blue Green Red
BRIEF	Binary Robust Independent Elementary Features
CNN	Convolutional Neural Network
CSP	Cross Stage Partial Network
CSV	Comma Separated Values
DFM	Design for manufacturing
DUT	Device Under Test
FCN	Fully Convolutional Network
FLANN	Fast Library for Approximate Nearest Neighbours
FPN	Feature Pyramid Network
GPS	Global Positioning System
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDI	High-Density Interconnect
HSV	Hue Saturation Value
IC	Integrated Circuit
ICT	In-Circuit Test
IDE	Integrated Development Environment
MLDN	Multisensor Lightweight Detection Network
NORMED	Normalised
OBB	Oriented Bounding Box
ORB	Oriented FAST and Rotated BRIEF
OTSU	Otsu’s Method
PCB	Printed Circuit Board
PIL	Python Imaging Library
PWB	Printed Wiring Board
PKL	Pickle
R-CNN	Region-based Convolutional Neural Network
RGB	Red Green Blue
SIFT	Scale-Invariant Feature Transform
SSD	Single Shot MultiBox Detector
SURF	Speeded-Up Robust Features
SQDIFF	Sum of Squared Differences
THRESH	Threshold
TPU	Tensor Processing Unit
YOLO	You Only Look Once
YOLOX	A version of YOLO

Berliner
Hochschule
für Technik

Chapter 1: Introduction

1.1 Motivation

Improving inspection processes in manufacturing provides a company with a competitive edge. With the introduction of new technologies and today's fast-paced world, organisations should keep up with what is new and adopt innovative solutions to maintain or expand their share in the market. Utilising digital image processing techniques in inspection processes has many benefits that cannot be overlooked.

Introducing automated image processing techniques into the manufacturing or assembly lines reduces operational costs and minimises waste. Additionally, it automates repetitive tasks, empowering the employee by saving time for more productive tasks with higher cognitive demands. Moreover, this also helps circumvent health issues associated with manual inspections, such as dry eye, eye strain and the risk of headache [25].

This proposes developing an automated inspection system using traditional and AI tools. This system is designed to reduce waste, such as inspection time, enhance productivity, and safeguard employee well-being by reducing the potential for injury. Furthermore, the system will be built to be flexible and versatile, ensuring that it can be used with various types of boards and is not exclusive or limited to a specific model or design.

1.2 Objective

This thesis aims to evaluate and compare traditional inspection techniques with advanced AI tools, specifically the YOLO algorithm, for inspecting PCBs. We will also explore and assess inspection methods and the latest AI technology that can be used to achieve that goal. By the conclusion of this thesis, both traditional and AI methods will be analysed. In addition, the result of combining both techniques will also be explored.

To accomplish these goals, the following steps were taken:

- Identifying and defining the types of defects expected in this challenge.
- Design and develop a system capable of establishing and defining a 'golden board' reference for each type of board to be inspected.
- Training AI models for the inspection process.
- Creation of a traditional and AI-based inspection pipeline for each method.
- Develop a system that integrates the abovementioned pipeline and visualises results.

1.3 Research Objectives and Questions

This thesis will explore and compare traditional image processing techniques with advanced CNNs to develop a more efficient and accurate system for automatic visual inspection of assembled circuit boards. The research questions include:

- How can CNNs improve the accuracy of PCB inspections compared to traditional methods?
- What are the limitations of current AI-based inspection systems, and how can they be overcome?

1.4 Significance of the Study

The research in this thesis will contribute to the field of electronics manufacturing by enhancing the efficiency and reliability of PCB inspections. I hypothesise that it can reduce costs, improve product quality, and speed manufacturing. This study will also provide insights into integrating AI technologies into manufacturing infrastructures.

1.5 Collaboration and Work Division

This thesis was completed in collaboration with Helen Dgheim. The division of work is as follows:

Table 1.1: Work Division

Task	**Contributor**
Program Front-End	Aiman Shahin
Program Back-End	Helen Dgheim and Aiman Shahin
Program Data-Structure	Aiman Shahin
PCB Board Alignment Method	Aiman Shahin
Augmented AI Dataset Creation and Labelling	Helen Dgheim and Aiman Shahin
AI Training	Helen Dgheim
AI in Software Integration	Helen Dgheim and Aiman Shahin
Inspection Method 1: Traditional Inspection	Aiman Shahin
Inspection Method 2: YOLO AI Inspection	Helen Dgheim
Inspection Method 3: Combined Method	Helen Dgheim and Aiman Shahin

Chapter 2: History and Literature Review

2.1 Printed Circuit Board

PCB is an essential component in electronics, serving as the foundation for connecting and supporting electronic components through conductive pathways [26]. These pathways are etched from copper sheets laminated onto a non-conductive substrate. PCBs are integral to various electronic devices, from simple digital clocks to complex industrial machinery [27].

2.1.1 History of PCBs

Early Developments

- Prior to the invention of PCBs, electrical and electronic circuits were manually wired point-to-point on a sheet metal frame or pan. These early assemblies were heavy, prone to failure, and costly to produce [28].

- In 1903, German inventor Albert Hanson filed the first patent for a device resembling modern PCBs. This invention, intended for telephone systems, featured flat foil conductors on an insulating board with multiple layers, through-hole construction, and conductors on both sides [29] [30].

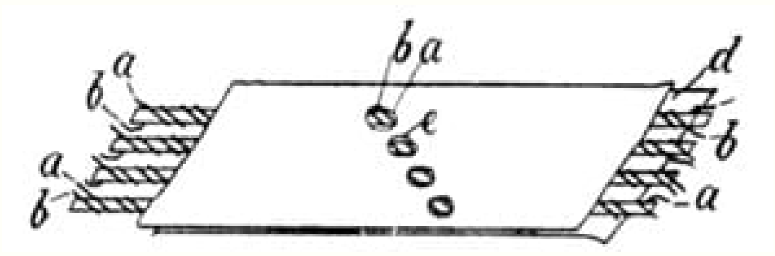


Figure 2.1: The First Albert Hanson PCB patent drawing
source: [29]

- Charles Ducas, in 1925, advanced the development of PCBs by patenting a method of electroplating circuit patterns. This technique, which involved applying conductive inks to an insulating substrate, marked the beginning of what would become the printed wiring board (PWB) [29] [30] [31].

Impact of Socio-Economic Events

- The evolution of PCBs experienced a significant slowdown during the Great Depression as economic hardships curtailed spending on what was then considered luxury electronic items [29] [30].
- World War II catalysed technological advancements, notably in communication infrastructures. The proximity fuse, developed by the British and enhanced by the US military for mass production, represented an early military application of PCB technology [29] [30] [31].

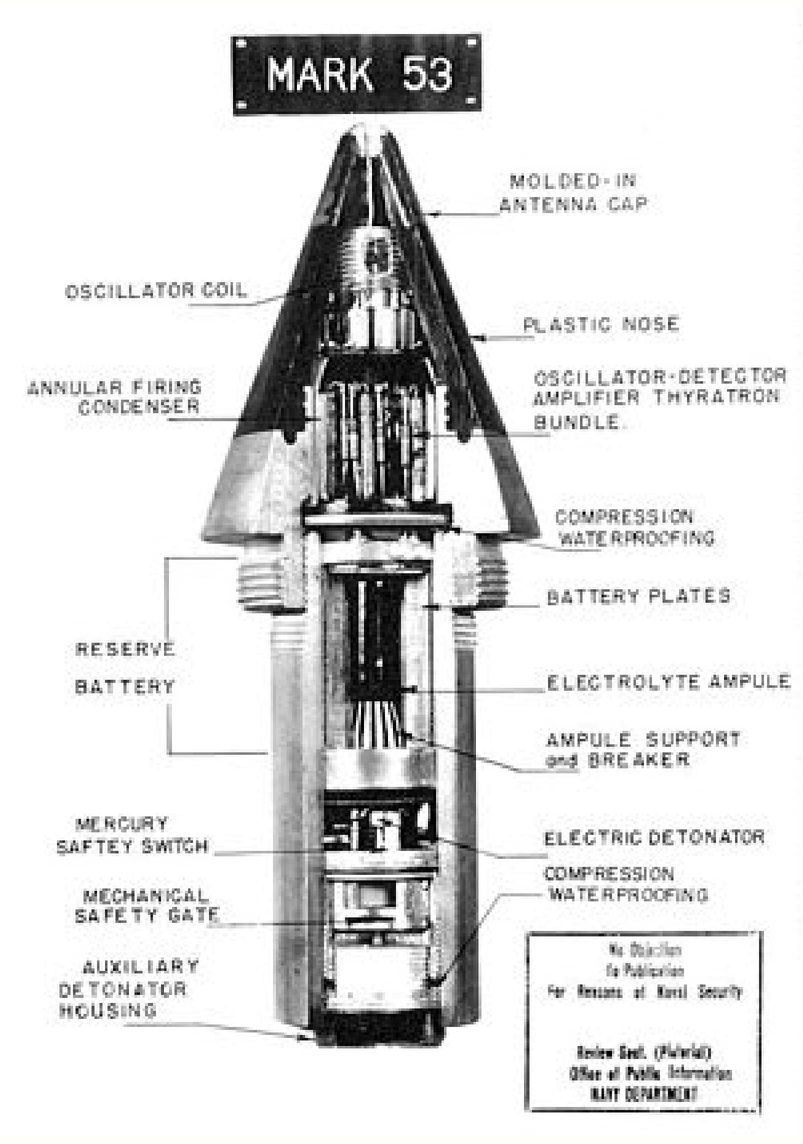

MARK 53
MOLDED-IN ANTENNA CAP
OSCILLATOR COIL
PLASTIC NOSE
ANNULAR FIRING CONDENSER
OSCILLATOR-DETECTOR AMPLIFIER THYRATRON BUNDLE
COMPRESSION WATERPROOFING
BATTERY PLATES
RESERVE BATTERY
ELECTROLYTE AMPULE
AMPULE SUPPORT and BREAKER
MERCURY SAFTEY SWITCH
ELECTRIC DETONATOR
COMPRESSION WATERPROOFING
MECHANICAL SAFETY GATE
AUXILIARY DETONATOR HOUSING
No Objection To Publication For Reasons of Naval Security
Review Sect. (Pictorial) Office of Public Information NAVY DEPARTMENT

Figure 2.2: Proximity fuse was the first military application to use a PCB
source: [29]

Post-War Developments and Commercialisation

- The post-World War II era and the ensuing Cold War saw significant advances in communication technology, beginning with the integration of Bell Labs transistors into PCBs [29] [30] [31].

- In 1956, the US Army's patent for the ”Process of Assembling Electrical Circuits” facilitated the creation of copper-trace components, which further advanced PCB technology [29] [30] [31].
- Motorola, in 1952, and Hallicrafters soon thereafter, were among the first to incorporate plated circuits into consumer electronics, marking the beginning of widespread PCB usage in products such as radios and clock radios [29] [30].

Technological Evolution in the Late 20th Century

- The transition from point-to-point chassis construction to PCBs in the late 1960s was driven by their reduced size, weight, and cost when integrated circuits (IC) were introduced in electronic designs [29] [30] [31].
- The 1970s witnessed the introduction of the microprocessor, exemplified by Jack Kilby's invention and the emergence of home computers like the MITS Altair 8800 and Apple 1 [29] [30].
- The 1980s saw electronics shrink further with the proliferation of devices like Walkmans, VHS players, cordless phones, and gaming consoles, all of which relied on PCBs for functionality [29] [30].
- The 1990s brought significant advancements in PCB design, particularly with the introduction of multi-layer surface boards that allowed for greater integration and reduced electronic interference. Computers became a staple in homes during this decade [29] [30] [31].
- In 1995, the development of microvia technology led to the production of High-Density Interconnect (HDI) PCBs, which enhanced signal speed and reduced electronic noise [29] [30].

Modern Advances

- Today, PCBs are integral to a wide array of devices essential to daily life, such as smartphones, computers, and household appliances, serving as the backbone of modern electronics [29] [30].
- In addition, what keeps driving PCB development is to keep up with technological trends and the necessity to develop better Internet of Things, AI, Automation and much more [28].

2.1.2 PCBs Design Variations

The Design of a PCB is dictated primarily by its intended application. Simple devices, for instance, mainly utilise single-layer PCBs with electrical components and conductive elements on one side of the board. In contrast, more complex devices, such as computer graphics cards, require multi-layer PCBs that accommodate their enhanced functionality and connectivity needs [27].

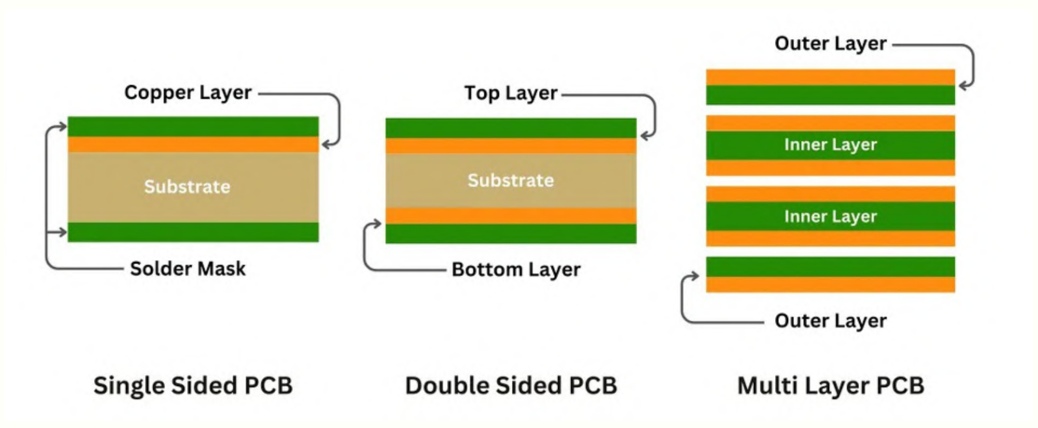


Figure 2.3: Comparison of single-layer, double-layer, and multi-layer PCBs
source: [27]

Single-sided PCBs

Single-sided PCBs have conductive traces only from one side of the board. They are commonly used for less complex circuit design since they are simple and cost-effective. However, due to their simple design, they pose substantial design challenges. They are limited in their routing options, challenging hardware design engineers to find the optimum route and try to integrate the advanced applications without compromising performance [27] [32].

Double-sided PCBs

Double-sided PCBs have conductive traces from both sides. Both sides are connected through vias holes. These holes are plated with conductive material, allowing interconnections between the layers. Double-layered PCBs provide more flexible opportunities to create complex routing that accommodates a higher component density. One additional advantage of double-layered PCBs over single-sided PCBs is their better thermal management, essential for maintaining component integrity and function under thermal stress [27] [32].

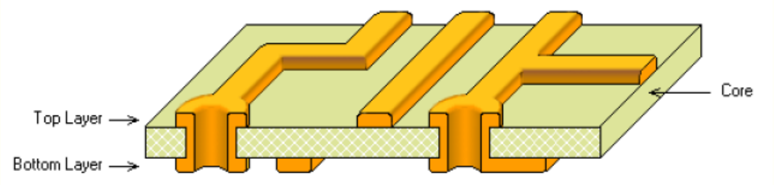


Figure 2.4: Double-Sided PCB
source: [33]

Multi-layer PCBs

Multi-layer PCBs consist of three or more conductive layers. Each layer is separated from the other with insulating material or prepreg. As mentioned before, the more layers, the more flexibility it offers and the higher the component density. This makes it suitable for advanced applications that require high-speed data transmission and processing and a high level of complexity, such as aerospace electronics. The addition of multiple layers facilitates not only enhanced functionality but also improved signal integrity, thermal management and noise reduction [27] [32].

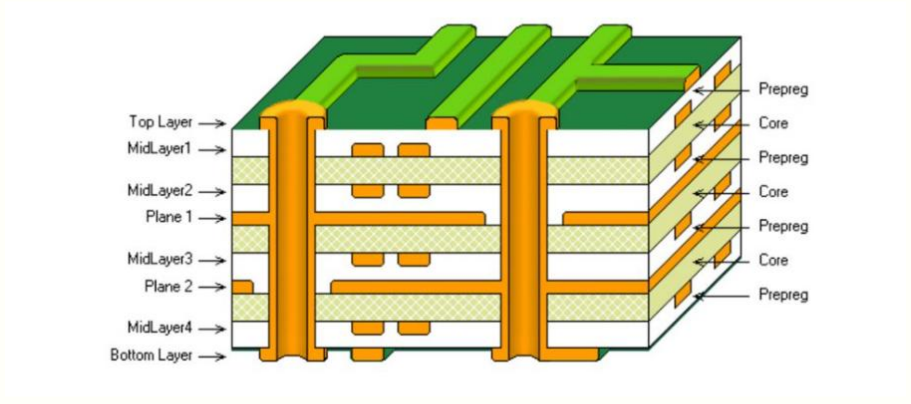


Figure 2.5: Multi-Layered PCB
source: [32]

To illustrate the differences among PCB configurations, consider an analogy comparing suburban living to residing in a high-rise building. In a suburban setting, travelling between distant buildings can be time-consuming, requiring vehicular or pedestrian transit across potentially expansive distances. This scenario parallels the challenges of routing on a single-layered PCB, where electrical connections must navigate around each other across the same plane, potentially leading to congestion and increased travel time for signals.

Conversely, a multi-layered PCB can be likened to a high-rise building with multiple elevators and staircases that provide direct routes between floors. This architecture significantly reduces the travel time between points by stacking critical components vertically and providing numerous pathways through vias, enhancing the efficiency of electrical connections. This configuration is particularly advantageous for densely packed circuits, where space optimisation and signal integrity are essential.

Therefore, in designing systems, it is crucial to consider the intended application's complexity and the need for efficient signal routing. Single-layered PCBs may suffice for more straightforward, less dense applications with manageable synchronisation and routing congestion. In contrast, more complex systems, where rapid and numerous interconnections are essential, typically require the sophisticated architecture of multi-layered PCBs.

Table 2.1: Characteristics Comparison of Different Types of PCBs

Characteristic	**Layers**	**Component Density**	**Complexity**	**Cost**	**Signal Integrity**	**Applications**
Single-sided PCBs	1	Low	Simple	Low	Low	Simple devices
Double-sided PCBs	2	Medium	Moderate	Medium	Medium	Consumer electronics
Multi-layer PCBs	≥ 3	High	High	High	High	High performance systems

source[27]

The complexity of a system, especially one that requires quick response and has numerous components, generally necessitates the use of multi-layered PCBs. This allows for more planned designs that can handle increased demands on the system.

Design Considerations

Considerations in PCB design encompass a variety of critical factors, each contributing to the functionality and reliability of the final product:

- **Component Placement:** Strategic arrangement of components to optimise board space and enhance performance [27].
- **Signal Integrity:** Ensuring clear and accurate signal transmission across the circuit without interference [27].
- **Thermal Management:** The board must be designed to manage heat generation effectively, which is crucial for maintaining component reliability and overall system stability [27].
- **Power Distribution:** Efficiently routing power throughout the board to ensure consistent operation under varying loads [27].
- **Manufacturing Constraints:** Accounting for limitations in fabrication and assembly processes that may affect the PCB design [27].
- **Cost:** Balancing design complexity with budgetary constraints to achieve cost-effectiveness [27].
- **Electromagnetic Compatibility:** Designing the PCB to minimise electromagnetic interference, ensuring compliance with regulatory standards and enhancing performance in complex environments [27].

2.1.3 PCB Components and Materials

PCBs are constructed from multiple layers and materials that contribute to their functionality and performance in electronic devices:

1. **Substrate:** The base foundational material of the PCB is usually made of fibreglass (FR4 is the most common). The substrate has good mechanical and electrical properties. Other types, like polyimide and PTFE, are used for high-frequency and high-temperature applications [27].
2. **Copper Layer:** This is the layer of copper foil laminated to the substrate where the electronic signals travel. In multi-layer boards, several copper layers are separated by insulation material [27].
3. **Solder Mask:** The solder mask is the layer on top of the copper foil. It insulates the copper traces to help prevent contact with other conductive materials or metals. This layer is typically green but can be seen in different colours, such as red, blue, or black [27].

4. **Silkscreen:** The silkscreen adds labels to the PCB surface to identify various components, test points, or logos. It is typically printed on the component side of the board and can be crucial for the device's assembly, inspection, maintenance, and repair [27].

5. **Surface Finish:** The surface finish is applied to the copper tracks exposed on the PCB to protect them from oxidation and improve the solder-ability of the copper during assembly [27].

6. **Components:** Components are the active (like transistors, ICs) and passive (like resistors, capacitors) elements soldered onto the PCB. These functional elements allow the electronic device to perform its designated functions when interconnected through the PCB's conductive pathways [27].

7. **Vias:** Vias connect the layers of a multi-layer PCB through small holes filled or plated with metal. Vias can be through-hole vias, blind vias, or buried vias, depending on whether they connect the surface layer to an inner layer or pass between layers internally [27].

8. **Pads:** Pads are small metal surfaces to which components' pins are soldered. They are crucial for forming reliable electrical connections [27].

2.2 PCB Defects

As with any other manufacturing process, defects occur in PCB manufacturing and assembly. This section will cover the types of defects that arise in PCBs, which were discussed in the following paper [1].

Table 2.2: Common PCB Defects

Defect	**Description**
Gaps in Solder Joints	Result from insufficient solder paste, component misalignment, thermal shock, or vibrations.
Solder Balling	Caused by impurities in solder paste or excessive heat.
Cold Solder Joints	Occur when solder does not reach the required temperature, insufficient flux, or prolonged thermal exposure.
Solder Bridging	Excess solder connects two conductors, often due to excessive solder paste or component misalignment.
Component Shift	Caused by improper positioning or excessive thermal energy during soldering. Detected via visual inspection and automated testing.
Lifted Pads	Occur when the solder mask or copper pad detaches from the PCB substrate due to thermal shock, vibration, or faulty components. Detected using AOI.
Webbing and Splashes	Caused by contaminants, leading to slender solder links (webbing) or excess solder dispersing (splashes). This results from substandard solder paste, inadequate cleaning, or insufficient flux.
Sunken Joints	Result from insufficient solder filling the gap between components and PCB pads, often during wave soldering.
Tombstoning	Occurs when discrete components stand upright on one end during wave soldering due to thermal imbalances.
Shadowing	Happens when surface-mount components do not fully contact the solder during wave soldering, leading to weak connections.
Opens	Absence of a bond between the lead and pad, resulting in an incomplete connection.
Excessive Solder	Leads to large solder bubbles at the joint, potentially concealing errors.
Missing Solder	Occurs when solder paste is absent on a PCB pad.

Defect	**Description**
Solder Short	Arises when components are inadvertently connected by excess solder paste.
Insufficient Solder	Refers to an inadequate quantity of solder on the PCB pad.
Missing Component	The absence of a required component on the PCB.
Tilted Component	Arises due to incorrect nozzle selection or blockage during assembly.
Lifted IC Pin	Occurs when the IC lead does not solder to the PCB pad, often due to improperly secured components.
Upside Down Component	Happens when a component with polarity is incorrectly flipped during placement.
Incorrect Component	Arises from selecting an unsuitable component during assembly.
Surface Scratches	Presence of abrasions on the PCB's surface.
PCB Bend or Warp	Condition is when the PCB is not flat and exhibits bends or distortions.
Overheat Blister	Occurs when the PCB and components are damaged due to excessive heat.
Poor Print Alignment	Arises when electronic components are inaccurately positioned or misaligned during assembly, leading to electrical and mechanical problems.
Scooping	Usually occurs during the soldering process due to issues with the stencil used to apply solder paste.
Solder Peaking	Characterised by the formation of elevated or peaked solder joints.
Solder Bleeding	Unintended spreading of solder beyond the designated joints or pads, causing solder bridges or shorts.
Solder Slump	Involves unintended displacement of molten solder from its intended location.
Head-in-Pillow	Defect where solder on both the PCB and component forms separately, resulting in two solder areas that fail to merge correctly.
Non-Wet Open	Occurs when solder paste initially on the circuit board pad is drawn onto the component side, resulting in an incomplete connection.
Solder Flags or Spikes	Result from inconsistent flux application and improper separation from the solder wave.
Cracked Joint	Occurs on a single-sided board where the solder joint fails due to expansion and contraction of the lead within the joint.

Defect	**Description**
Flux Residues	Excessive or poorly controlled flux residues can cause various issues despite enhancing solder adhesion when appropriately applied.

2.3 PCB Inspection Methods

It is crucial to create a PCB inspection system suitable for the manufacturing process to find these defects as they will mainly produce malfunctions, which could lead to damage. Depending on the manufacturing stage, a particular type of inspection is used. However, this is also limited to the equipment available. In the following section, different types of inspection methods will be covered.

2.3.1 Manual Inspection

Using a magnifying glass, a trained technician manually examines the PCB. This method represents one of the earliest approaches to visual inspection technology, which has evolved significantly to enhance production, improving efficiency according to Caixia Wu [2] and supported by Singh et al. [1].

In addition, Singh et al. [1] mention that manual visual inspection is used for soldering defects, misaligned components, and other critical visual abnormalities, improving production efficiency. However, they warn that prolonged manual inspection needs to be improved in reaching inspection goals. This method is recommended for components that require stringent quality control to meet 100% of the expected standards.

Figure 2.6: PCB Circuit Board Manual Inspection
source: [34]

2.3.2 Automated Optical Inspection

In an article by the Matric Group, a group that provides top-notch Electronic Manufacturing Services (EMS), [35] AOI captures a PCB photo using either a single 2-dimensional (2D) camera or two Three-dimensional (3D) cameras. These photos are then compared to a golden board reference containing a detailed schematic, Flagging a board that does not match the schematic. The article also mentions that AOI is best used to detect issues early, shutting down production. It is also recommended that this type of inspection be coupled with other inspection methods as it cannot be used solely for inspection.

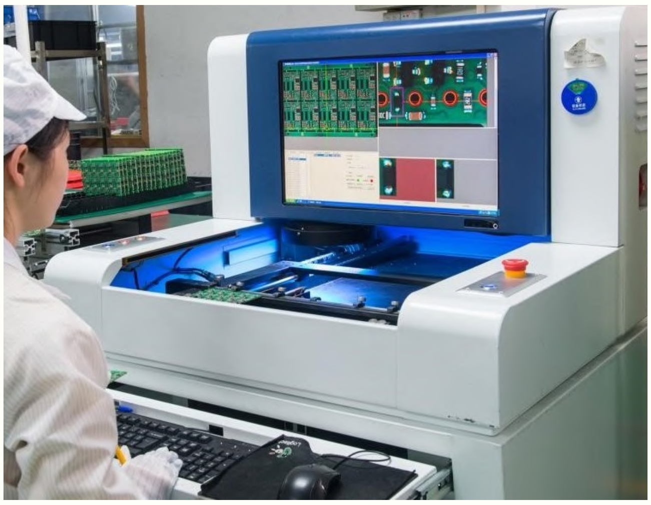


Figure 2.7: PCB Circuit Board Automated Optical Inspection
source: [36]

The following study done by Ebayyeh et al. [3] provides critical considerations for using AOI:

- **System Calibration and Configuration:** Precise calibration is crucial for accuracy. Otherwise, incorrect calibration limits the system's adaptability to different products.
- **Illumination Settings:** The correct setup helps avoid shadows and light noise, making it easier to filter the image. Depending on the defect being inspected, illumination can be adjusted for that specific defect.
- **Camera and Lens Selection:** Depending on the working distance, captured resolution, depth of field and field of view, camera positioning and lens choice are determined. That is, to ensure the capture of high-quality images for accurate inspection.
- **Inspection Algorithms:** The correct algorithm should be chosen according to the setup and inspected defect. Generally, the inspection involves three stages: pre-processing, feature extraction and selection, and classification. The study also mentions that CNNs are coming into trend due to their ability for classification and feature extraction.
- **Environmental and Operational Conditions:** Variable illumination must be avoided to avoid errors and wrong inspection results. Moreover, the system should handle environmental changes.
- **Data Requirements for Machine Learning:** Using machine learning with AOI requires many training samples. That is why industries must create as many relevant images as possible.
- **Limitations and Future Directions:** The study mentions that the AOI system has difficulties inspecting internal defects and needs a large dataset for machine learning models. In addition, the study suggested that the system needs improvements to handle environmental changes.

2.3.3 X-ray Inspection

The Matric Group article [35] defines X-ray inspection, also known as Automated X-ray Inspection (AXI), as a tool rather than an inspection method that combines 2D and 3D. The main advantage of X-ray testing is that it detects various types of defects that are often invisible to optical inspection methods. However, the article also mentions that this method requires trained and experienced operators. Another setback is that it is time-consuming and expensive.

C. Neubauer mentions in this study [4] that another advantage of the X-ray method is that it provides high-accuracy results in detecting solder thickness, volume and voids that are hard to analyse using visual inspection. It can also inspect solder joints like ball grid arrays (BGAs) and chip-on-board components. It also enables detailed profiling of solder joints, such as wetting angles, distribution, and alignment between pad and lead. However, the X-ray inspection method requires a complex setup and precise calibration and alignment of X-ray sources and detectors. If the board has high density, the inspection process will be slow.

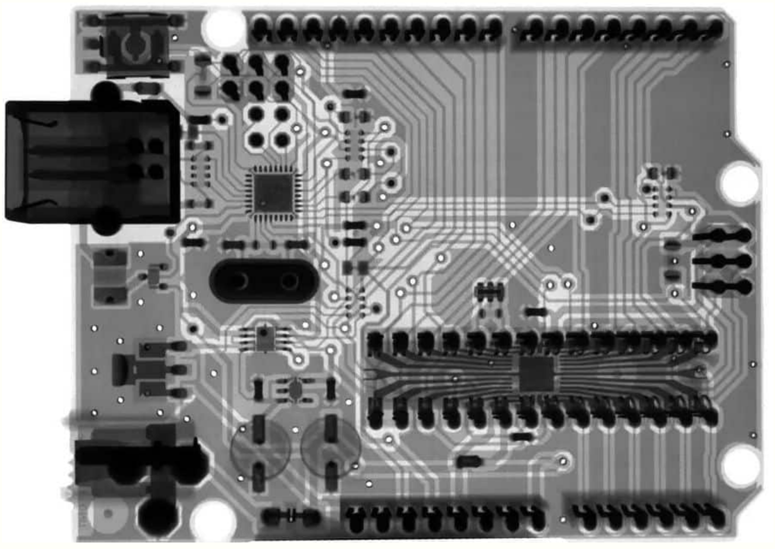


Figure 2.8: PCB Circuit Board Xray view
source: [37]

The study [4] also mentions the critical considerations when using the X-ray method:

- **System Calibration and Setup:** X-ray detectors should be precisely calibrated to achieve high-resolution and accurate imaging and avoid image distortion.
- **Handling Complex Board Designs:** For double-sided PCBs, it is better to consider using 3D reconstruction techniques such as laminography or microtomography to separate superimposed components and obtain clear images.
- **Speed vs. Accuracy Trade-off:** 2D inspections are better used for rapid evaluation, while 3D inspections are used for more complex situations.

- **Safety and Environmental Considerations:** X-rays cause radiation exposure; therefore, it is necessary to comply with safety regulations and protocols. In addition, X-ray tubes and detectors need to be disposed of in a safe manner.

2.3.4 In-Circuit Testing

According to the article made by the Matric Group [35], In-circuit testing (ICT), also known as the ”bed-of-nails” test, is one of the most robust methods for testing PCBs. It is expensive, with costs influenced by the board and fixture size. ICT powers up and actuates individual circuits on the PCB, aiming for 100% coverage but typically achieving about 85-90%, free from human error.

The process uses fixed probes arranged to match the PCB design. These probes check the integrity of solder connections. The test starts when the PCB is pressed onto the bed of probes, with predesigned access points allowing connections to the circuitry. ICT is effective for larger connections and ball grid arrays and is best suited for mature products with few expected revisions. It requires design-for-manufacturing (DFM) principles from the beginning, including proper test pads.

The following study [5] Schaaijk et al. support the article regarding the good coverage of the ICT in testing individual components and connections on the PCB, often aiming for near or total coverage. The study also adds that it is possible to inspect the board without powering it up, preventing potential damage and identifying faults in passive components before any power-up. ICT can also detect multiple defect types like presence, correctness, orientation, liveliness, and alignment of components, as well as shorts, opens, and quality of connections. However, setting up ICT and the equipment itself is expensive. In addition, ICT depends on the accessibility of probe points. Like in surface mount technology, it can pose a challenge to access their leads. ICT might also face challenges when testing boards with mixed technologies like through-hole technology and surface mount technology.

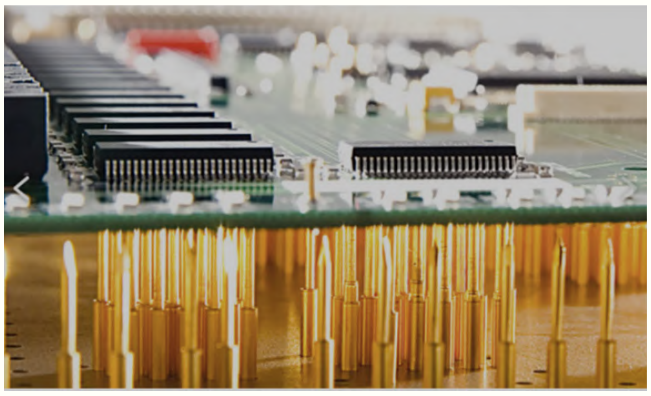


Figure 2.9: PCB Circuit Board In-Circuit Probes
source: [38]

Schaaijk et al. [5] also mention the critical considerations when using the ICT method:

- **Design for Testability:** PCB design should include probe points for accessibility and should have a proper layout to facilitate ICT.

- **Fixture Design:** The bed-of-nails fixture must be precisely designed to match the PCB layout.

- **Impedance Handling:** An impedance graph needs to be created according to the PCB impedance characteristics to represent the electrical properties and identify false paths that may affect the test results.
- **Guarding and Path Neutralisation:** additional probes should be used to neutralise false paths and ensure accurate testing of the target components
- **Voltage Limitations:** The maximum voltage allowed for testing the components should be taken into consideration to avoid damaging it.

2.3.5 Flying Probe Testing

The Matric Group article [35] mentions Flying probe testing as a cost-effective alternative to ICT. This non-powered method checks for opens, shorts, resistance, capacitance, inductance, and diode issues using needles attached to a probe that moves along an x-y grid guided by CAD-based coordinates.

Although flying probe testing is initially cheaper and suitable for small production runs or prototypes, ICT, despite its higher initial costs due to custom test fixtures, is faster and less error-prone for large-scale production. The choice between these methods depends on production volume and cost considerations. Notably, flying probe testing does not power up the PCB during testing, making it less invasive but potentially less comprehensive than ICT.

According to the following article [39] written by Muhammad Sufyan, the Flying probe inspection method is easily adaptable to different PCB designs without needing a custom fixture. That is why it is also cost-effective while maintaining the high coverage of ICT. The method of inspection minimises contact with the PCB, reducing damage risk. However, unlike the ICT, it is slower due to the physical movement of the probes. The probes will have limited access to some test points if the board is dense or complex.

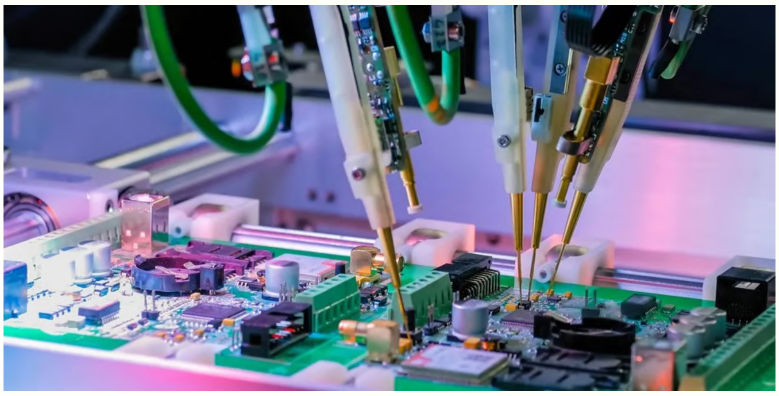


Figure 2.10: Flying Probe Testing
source: [39]

Muhammad Sufyan [39] also mentions the critical considerations when using the flying probe method:

- **Regular Maintenance:** Essential for maintaining probe accuracy and system reliability.

- **optimised Test Programs:** It is essential to optimise probe paths and test steps for efficiency.
- **Fixture Design:** Ensuring secure PCB placement to prevent false readings.
- **Integration:** Effective integration with other manufacturing processes to streamline production.
- **Continuous Monitoring:** Regular analysis and improvements to maintain high-quality testing standards.

2.3.6 Functional Testing

The following white paper [6], made by Cal Houdek, an Engineer and Owner at Caltronics Design & Assembly Inc., provided an overview of different types of PCB inspections. One of them is Functional testing, which is a crucial method for assessing a PCB's performance by replicating its final electrical environment. This process involves interfacing with the PCB through its edge connector or test-probe points, simulating the conditions the PCB will face in its actual application, such as a computer slot.

The White paper [6] finds that functional testing can detect issues that affect the PCB's operational capabilities. Since it simulates a final electrical environment, it can accurately measure the power usage of the device under test (DUT) during operation. This test can uncover issues like analogue and digital circuit problems. However, due to the complexity of programming, it requires a significant investment.

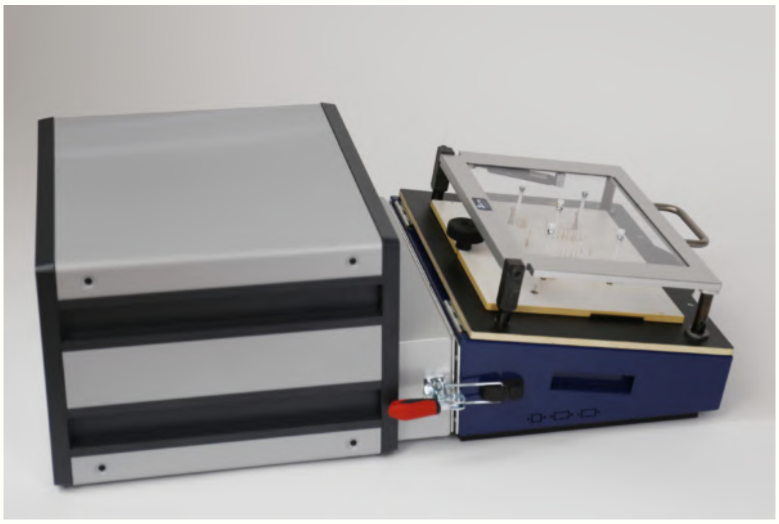


Figure 2.11: Functional Tester with Connected Test Adapter
source: [40]

The white paper [6] also covers the key considerations when inspecting using functional testing:

- **Customisation:** Each functional tester is often tailored to the specific DUT, necessitating detailed knowledge of the DUT and its environment.
- **Comprehensive System:** The tester system typically includes a cabinet, DUT interface, cabling, a CPU, and monitors, with hardware varying based on the DUT's requirements.

- **Complexity and Variability:** Functional testing involves numerous variables, including the extent of PCB testing, necessary inputs, desired outcomes, and test parameters, contributing to its complexity.

2.3.7 AI use in PCB Inspection

Traditional defect detection methods such as manual inspection, performance testing, and AOI have limitations in accuracy and efficiency. Therefore, in this study, [7], Chen et al. have introduced utilising YOLO models as an effective solution for PCB defect detection. Multiple other models can also be used, such as Neighbourhood Correlation Enhancement Networks, Fourier Image Reconstruction, Fully Convolutions Networks, and more. All have the advantage of better classifying the types of defects than traditional methods. However, one of the significant drawbacks of using AI and machine learning in PCB defect detection is the scarcity of samples available to train the model.

2.4 Conventional Algorithms for PCB Defect Detection

As mentioned before, AOI uses algorithms and techniques to compare board components with a golden reference. In this section, we will explore different algorithms that are used in PCB inspection, such as Speeded-Up Robust Features (SURF), Oriented FAST and Rotated BRIEF (ORB), Binary Robust Independent Elementary Features (BRIEF), and Scale-Invariant Feature Transform (SIFT). The comparison focuses on their effectiveness, success rates, and specific applications within PCB inspection.

2.4.1 Scale-Invariant Feature Transform

SIFT is known for its ability to detect and describe local features that are not affected by scale or rotation. It is also quite effective in environments with changing illumination and perspective. Studies like the one done by Rehman et al. [8] mentioned that SIFT has better feature extraction than SURF and demonstrates high accuracy in feature detection and matching. However, they also mention that it is computationally intensive, which might make it slower compared to other algorithms.

2.4.2 Speeded-Up Robust Features

SURF is an enhancement of SIFT, where it was made to be faster while maintaining similar performance as referred to in this document [41]. SURF rely on the determinant of the Hessian blob detector, which makes it computationally less intensive. It is also suitable for real-time application. However, Rehman et al. [8] mention that SURF does not detect features as well as SIFT. This document [41] also mentions that SURF is better in blurred images and rotation but is not as good at viewpoint change and illumination change. This study by Hassanin et al. [9] found that SURF demonstrated efficient automatic defect detection in PCBs.

2.4.3 Oriented FAST and Rotated BRIEF

ORB is an algorithm developed in OpenCV-Labs; according to this document from OpenCV [42], ORB was made for the purpose of being a more efficient and faster algorithm than SIFT and SURF. In addition, it is not patented, unlike SURF and SIFT. The document mentions that ORB is faster than both SIFT and SURF and is better at feature extraction than SURF. It combines features from the Accelerated Segment Test (FAST) keypoint detector and the BRIEF descriptor. ORB is usually used when low computational power is available.

Binary Robust Independent Elementary Features

The following OpenCV document [43] mentions that BRIEF is a binary descriptor, which means that it needs to be used with other feature detectors like SIFT, SURF and ORB; although invariant to scale and rotation, it is very fast and suitable for applications where speed is essential, and conditions are maintained constant.

Features from Accelerated Segment Test

Similar to BRIEF, FAST is a keypoint detector used for corner detection, as mentioned in the following article [44]. FAST is used in combination with other descriptors like BRIEF, as it alone is not enough for complete feature matching.

Table 2.3: Comparative Analysis of SIFT, SURF, and ORB Algorithms for PCB Inspection

Algorithm	**Advantages**	**Disadvantages**	**Suitable Applications**
SIFT	High accuracy, robust to scale/rotation	Computationally intensive	Detailed feature matching, complex scenes
SURF	Faster than SIFT, good accuracy	Slightly less accurate than SIFT	Real-time applications, fast inspections
ORB	Very fast, low computational cost	Less accurate than SIFT and SURF	Resource-constrained environments

2.4.4 Template Matching Algorithm

The following document [45] explains that the template matching process involves sliding a template image over an input image and calculating a similarity measure for each position to identify where the template matches the input image. Template matching is very fast and suitable for environments where the illumination does not change. However, it has problems in rotation and scale. But this step can be overcome by more pre-processing of the inspected elements by either scaling up or down or rotating the input image to match the template in order to find a match like what was done here by Sassanapitak et al. [10], where they used template matching algorithm to match components that are rotated with the golden template, which provided excellent results performance for printed circuit board assembly (PCBA) inspection.

Template matching offers several advantages in PCB inspection:

- **Simplicity:** The method is straightforward to implement and understand, making it accessible for various applications in PCB inspection. [46]
- **Precision:** Template matching provides precise localisation of components, ensuring accurate alignment and placement verification. [46]
- **Integration:** It can be easily integrated with other techniques, such as genetic algorithms and deep learning, to enhance robustness and accuracy. [46]

2.4.5 Conventional Algorithm used in PCB inspection

From the previous section, it was determined that template matching is one of the simplest and fastest algorithms to use. However, due to its inability to scale up and down and having problems with rotation, it was decided to use one of the feature-matching algorithms. The SIFT algorithm was chosen because it is said that the SIFT algorithm has the best feature extraction algorithm, and it has good performance even in varying illumination and viewpoint more than SURF and ORB. In addition, like most feature extraction algorithms, it is invariant in scale and orientation. Although the SIFT has a more computational cost, this cost is acceptable as the primary method of inspection will be using a template matching technique, and when template matching fails, the SIFT method will be used.

Inspection Method Design

As mentioned before, the Template Matching algorithm will be the main algorithm used in the conventional inspection process for classifying and localising PCB components. In case the template matching technique fails, the SIFT algorithm will be used to match the component.

Preliminary Flow of Inspection

- Component will be matched with the golden template of the component, which is supposed to be located in the same location stored in the golden reference, using a template matching algorithm.
- If the golden template fails, the program will extract secondary golden templates of the same component type from the program database and try to match the component with the secondary template using the template matching technique.
- If the secondary templates fail, the program will then use the SIFT algorithm to match the component with the original golden template component.

2.5 AI Models for PCB Defect Detection

In a previous section, we discussed the limitations of traditional techniques in terms of accuracy and efficiency. This is why using AI to improve the quality of inspections yields better results [7]. In this section, we will explore and compare different AI models.

2.5.1 Yolo Models

Reviewing Singh et al. [1], the YOLO models have been found to be effective for PCB defect detection due to their high accuracy and speed in real-time inspection. Different versions of YOLO have been developed to tackle various challenges in object detection, including PCB defect detection. In this section, various YOLO models, such as YOLO v3, YOLOv5, and YOLOX, will be covered.

YOLO v3

Using convolutional layers, this model can directly predict from full images bounding boxes and class probabilities in a single evaluation. Li et al. improved the YOLOv3 accuracy by applying a joint approach of preprocessing data of real and virtual PCB images by changing the three-layer predicted output to a four-layer predicted output layer, demonstrating better stability and accuracy in detecting common PCB defects, achieving an accuracy of 93.07% [11].

YOLO v4

By incorporating new features such as the Cross Stage Partial (CSP) network, YOLOv4 enhances the architecture of YOLOv3, which improves learning capability and reduces memory costs [7]. Xin et al. improved the YOLOv4 network model by analysing the backbone architecture CSPDarknet53 and adjusting the model hyperparameters settings, increasing the accuracy from 89.60% to 96.88% [12].

YOLO v5

YOLOv5 has an improved detection performance. It maintains the overall architecture of YOLOv3 and YOLOv4. YOLOv5 introduces optimisation in the input processing, including data augmentation and anchor box learning, achieving higher accuracy and efficiency in detecting PCB defects [7].

Transformer-YOLO

Chen et al. proposed a new network model based on YOLOv5 called Transformer-YOLO, which enhances the original clustering algorithm to create anchor boxes better suited for the PCB defect dataset, resulting in improved detection accuracy. They also replaced the traditional CNN with the Swin Transformer as the main feature detection network. The Swin Transformer enhances the interaction between image features, providing a strong global mechanism. Additionally, it includes an attention mechanism to focus on high-importance channel information, achieving a detection accuracy of 97.04%. This outperforms Faster R-CNN, SSD, YOLOv3, YOLOv4, and YOLOv5 by margins of 23.90%, 15.51%, 10.70%, 7.83%, and 6.12%, respectively [7].

YOLOX-MC-CA

Xuan et al. created a new model based on the YOLOx framework called YOLOX-MC-CA. This model includes a modified CSPDarknet backbone and a Coordinate Attention mechanism to improve the detection of small defects on PCB surfaces. The backbone network of YOLOX is modified to include inverted residual blocks and depthwise separable convolutions, reducing computational overhead and memory usage while maintaining strong feature extraction capabilities. The CA mechanism enhances the network's ability to focus on the spatial information of small defects, which is crucial for accurate detection. This is achieved by encoding channel correlation and precise coordinate information, significantly improving detection precision for small defects. YOLOX-MC-CA Achieved the highest detection precision with an accuracy of 99.13% [13].

2.5.2 Other AI Models

Fully Convolutional Networks (FCN)

Improved Fully Convolutional Networks: Zheng et al. adopted the MobileNet-V2 networks model to improve CNNs. This approach has been used to detect four common PCB defects—spurs, mouse bites, short circuits, and open circuits—achieving a high classification accuracy of 92.86% [14].

Feature Pyramid Network (FPN) And Single Shot MultiBOX Detector (SSD)

FPN & SSD: Wu et al. trained FPN and SSD models on two different datasets, where they provided high accuracy for PCB defect detection. SSD achieves 98.9% accuracy on the PCB dataset, while FPN achieves 96.4% [15].

Neighborhood Correlation Enhancement Network (NCE-Net)

NCE-Net: Huang et al. introduce a groundbreaking strategy that utilises defect and surrounding relationship information to distinguish defect authenticity accurately. This method employs SRB and RRB modules and has achieved an accuracy of 92.59% [16].

Multisensor Lightweight Detection Network (MLDN)

MLDN: Li et al. used MLDN to find defects that usual traditional inspection methods have challenges finding by combining polarisation information with infrared and visible brightness intensities to detect defects traditional optical inspection methods may miss. This method has shown an impressive accuracy of 96.2% [17].

2.6 Rationale and Methodology for Using YOLO in PCB Quality Control

For this thesis, there are several compelling reasons for choosing YOLO models for PCB defect detection. First, as mentioned in the previous section, YOLO models are well-known for their high accuracy and speed in real-time object detection tasks, making them particularly suitable for the fast-paced environment of PCB manufacturing. The ability of YOLO models to process images in a single evaluation significantly reduces the time required for defect detection compared to traditional methods, which often involve multiple stages and higher computational costs.

The YOLO models, including YOLOv3, YOLOv4, YOLOv5, and YOLOX, are adaptable across different versions. This flexibility allows for fine-tuning to address specific challenges in detecting PCB defects. Each iteration of YOLO brings improvements in feature extraction, processing efficiency, and accuracy, ensuring that the latest models can effectively handle the intricate and varied nature of defects found on PCBs.

2.6.1 Approach Using YOLOv8 for Component Detection

The thesis proposes a different approach using YOLOv8 to detect PCB components rather than directly identifying defects. This strategy is based on PCB components' consistent shapes and sizes across different boards, making them suitable for object detection models like YOLOv8. In contrast, defects on PCBs can vary widely in appearance, and with a limited number of defects in the dataset, this presents a challenge for detection models.

By training YOLOv8 to recognise and locate standard components on a PCB, the model's ability can be used to identify these known entities accurately. Any deviations from the expected placement, alignment, or presence of these components can then be flagged as potential defects. This indirect defect detection method allows for a more reliable identification process, as the model focuses on detecting consistent and well-defined objects.

This approach also simplifies the training process by requiring a more straightforward dataset to train the model on component detection. This leads to faster convergence during training and potentially higher overall model accuracy.

By leveraging YOLOv8 for PCB defect detection through component recognition, the challenges posed by defects' diverse and irregular nature can be addressed, providing an efficient solution for ensuring PCB quality in manufacturing processes.

Table 2.4: Comparison of Defect Detection and Component Detection in PCB Inspection

Inspection Based On	**Defect Detection**	**Component Detection**
Variation	High	Low
Used In	PCB Manufacturing Stage	Component Assembly Stage
Dataset Required	Large	relatively smaller
Focus Area	Surface Defects, Soldering Errors	Component Placement, Orientation

Chapter 3: Tools and Methods

3.1 Development Environment

Specific development environments were used to build this project. Python was chosen due to its simplicity and strong support for tools like YOLO Ultralytics and Python Qt (PyQt), a graphical user interface-building tool. JetBrains PyCharm IDE was used to build the front and back end of the application, as it provides advanced support in coding. Google Colab were used to train the AI model, as Google Colab offers powerful tools that make the training faster.

3.1.1 Python

Python, known for its simplicity and readability, was released in 1991 by Guido van Rossum. It is a high-level programming language that is interpreted by novices and experienced programmers. Python's design philosophy uses significant indentation to enhance the readability of its code. Python supports multiple programming paradigms, making it flexible when developers choose their programming approach as it includes paradigms like procedural, object-oriented, and functional programming. Its extensive standard library and active community contribute to widespread adoption across various domains, like web development, software development and AI. The versatility of Python is evidenced by its use in numerous fields. Frameworks on the web, such as Django and Flask, help create strong and scalable web applications. In scientific computing, libraries like NumPy, SciPy, and pandas provide powerful tools for data analysis and numerical computations. Additionally, libraries such as TensorFlow and PyTorch support Python's prominence in AI, which enables the development of complex machine-learning models. The language's ease of learning and extensive ecosystem make Python a pivotal tool in modern software development and research. [47]

3.1.2 JetBrains PyCharm:

PyCharm, developed by JetBrains, is a powerful integrated development environment (IDE) tailored to Python programming. Renowned for its robust feature set, PyCharm provides tools for both professional developers and beginners. It supports various web development frameworks like Django and Flask, integrates with popular version control systems like Git, and offers advanced code editing features, including intelligent code completion, on-the-fly error checking, and quick fixes. PyCharm's extensive debugging and testing capabilities allow developers to write clean, maintainable code efficiently.

Moreover, PyCharm enhances productivity with its rich ecosystem of plugins and its seamless integration with other JetBrains tools. The IDE also supports scientific development with data science and machine learning tools, including integration with Jupyter notebooks and support for libraries like NumPy and pandas. These features make PyCharm a good tool for developing this project [48].

3.1.3 Google Colab

A cloud-based Jupyter Notebook allows Python code to be written from a web browser. It also has Graphics Processing Units (GPU) and Tensor Processing Units (TPU) computer resources that can be used for free for a limited time, eliminating the need to invest in a powerful computer. It is mostly used for machine learning and is connected to Google Drive, a cloud-based storage space that facilitates data access. [49]

3.2 Methodology

This section outlines the methodologies employed in this thesis, detailing where and how each is utilised within the workflow. The methodologies are presented in the order of their initial use, although some may recur throughout the process. Each methodology is subsequently defined and discussed in detail.

3.2.1 Overview of Methodologies

In this section, we will go through the methodologies used in this thesis, and later, each will be discussed in detail.

Trilateration: Trilateration is a technique used to verify the accurate location of components detected by the CNN or to predict the location of components pre-inspection for standard template-matching inspection. This method is a critical part of the inspection process and requires fine-tuning to ensure precise component localisation.

Noise Reduction and Filtering: Filtering operations, such as Gaussian filtering, are employed to remove noise from component images, either to save them as templates or to prepare them for inspection. This step is crucial because the inspection process is highly sensitive to changes, and filtering helps reduce this sensitivity by making similar components appear more alike. This process is used in the PCB preparation phase to create templates before inspection.

Binarization Techniques: After filtering, components are binarized to extract simpler shapes or text, which can then be saved as templates or used in other tools. This step occurs after component filtering in the PCB preparation phase, simplifying the components for further processing.

Morphological Operations: Morphological operations are applied to binarized images to remove noise and better define objects. These operations are a subset of the binarization process, enhancing the extraction of information from components by refining their shapes.

Edge Detection using Canny: Canny edge detection is an additional technique for extracting edges from filtered or binarized images, which are then saved as templates for later use. This method is employed after filtering and optionally after binarization during the PCB preparation phase.

Template Matching: Template matching is performed during inspection after preprocessing the PCB image. This technique compares components with stored template components (normal, grey, filtered, or grey-filtered) to identify matches. The preprocessing steps applied to the image during inspection mirror those used to create the template, ensuring accurate comparisons.

Sift Algorithm: If template matching fails to find a match, the SIFT algorithm is used as a secondary option. SIFT provides a higher matching accuracy and can detect component rotations, though it requires more processing time. Therefore, it is used as a backup during the inspection process.

CNN: As outlined in the thesis objectives, CNNs are embedded in the inspection pipeline between preprocessing and postprocessing. The CNN model identifies PCB components and provides their locations in the image. Postprocessing is then required to localise the components and evaluate the board for defects accurately.
In the following sections, each methodology will be discussed in more detail.

3.2.2 CNN

Convolutional Neural Networks are a class of machine learning specifically designed to process and analyse visual data. Inspired by the human visual system, they are effective in recognising patterns.[50] CNN models for object detection can be categorised as follows:

- **Single-Stage Detectors:** offer faster computational but lacks accuracy. [18]
- **Two-Stage Detectors:** propose regions of interest, then classify those regions and refine their location predictions. [18]

Single Stage CNN

YOLO, in addition to SSD, is an example of a single-stage CNN. These models do not propose regions so the network can search; instead, they directly localise and identify objects in a single step. This makes the detection process quick compared to two-stage methods, prioritising speed over accuracy. This is why they encounter difficulties when dealing with minor or closely positioned objects [18]. However, this is being tackled in new YOLO models, which will be discussed later. [51][52][53]

Double Stage CNN

Examples of double-stage CNN are Fast-RCNN, Faster-RCNN and MASK RCNN. As mentioned, double-stage CNN is a slow detection CNN; however, each tried to overcome this through different methods.[18]

- **Fast-RCNN:** Improves efficiency by processing the image through a CNN once to generate a convolutional feature map, reducing the convolution operations needed per image. [18]
- **Faster-RCNN:** Enhances speed by using a network to predict region proposals instead of the slower selective search, then applies Fast-RCNN techniques for object detection within these regions. [18]
- **Mask-RCNN:** Extends Faster-RCNN by adding a segmentation feature that creates a binary mask for each bounding box to distinguish object pixels, thereby increasing accuracy. [18]

3.2.3 Noise reduction and Filtering:

Noise reduction in images is essential to clear the image from elements that can affect the image processing operation. Several methods, such as Gaussian Filtering, Bilateral Filtering, and Fast Non-Local Means, are used to smooth the images.

Gaussian Filtering

Gaussian filters are commonly used for noise reduction and smoothing. They apply a Gaussian function to weigh the pixels within the filter window, resulting in a blurred version of the image that reduces high-frequency noise [54].

- **Mechanism**: The Gaussian filter is defined by the equation:

  $$G(x,y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

  Where $G(x,y)$ is the Gaussian function, and σ is the standard deviation, which controls the extent of the smoothing [54].
- **Application**: The filter is convolved with the image to produce the smoothed image. The larger the σ, the greater the smoothing effect, but excessive smoothing can lead to losing essential details [54].
- **Advantages**: Simple to implement and effective for reducing Gaussian noise [54].
- **Limitations**: Can blur edges and fine details of the image [54].

Bilateral Filtering

Bilateral filters (BF) preserve edges while smoothing images, making them useful for noise reduction without sacrificing important details [55].

- **Mechanism**: The BF combines spatial and intensity information:

$$BF[I]_{\mathrm{p}} = \frac{1}{W_p} \sum_{q \in S} G_{\sigma s}(\|p - q\|) \cdot G_{\sigma r}(\|I_p - I_q\|) \cdot I_q$$

  Where $BF[I]_{\mathrm{p}}$ is the filtered intensity, σ_s controls the spatial domain, σ_r controls the intensity range, and W_p is the normalisation factor [55].

- **Application**: The filter applies a weighted average based on the spatial distance and intensity difference, which helps maintain edges while reducing noise [55].

- **Advantages**: Excellent at preserving edges and fine details while reducing noise [55].

- **Limitations**: Computationally intensive, especially for large images [55].

Fast Non-Local Means Filtering

Fast non-local filters reduce noise by considering the entire image rather than just local neighbourhoods, thus preserving finer details and textures [19].

- **Mechanism**: These filters compare image patches rather than individual pixels. The similarity between patches is used to average the pixels, defined by:

$$I_{\mathrm{filtered}}(p) = \frac{1}{C(p)} \sum_{q \in \Omega} I(q) e^{-\frac{\|I(p)-I(q)\|^2}{h^2}}$$

  Where $I_{\mathrm{filtered}}(p)$ is the filtered pixel intensity at p, h is the filtering parameter, and $C(p)$ is the normalisation factor [19].

- **Application**: The filter computes the weighted average of similar patches within a particular search window, effectively reducing noise while preserving texture and detail [19].

- **Advantages**: Highly effective for denoising while maintaining fine details [19].

- **Limitations**: Requires significant computational resources, although optimised versions exist [19].

3.2.4 Edge Detection Canny

Canny edge detection is an established method in computer vision, introduced by John F. Canny in 1986. It is designed to identify object edges within an image characterised by significant changes in pixel intensity, indicating transitions between different objects or regions.

It is a multi-stage algorithm used in computer vision to detect a wide range of edges in images. It includes several key steps which involve mathematical computations to enhance accuracy and effectiveness:

1. **Gaussian Filtering:** The image is first smoothed using a Gaussian filter to reduce noise. This is represented mathematically as the convolution of the image I with a Gaussian kernel $G(x,y)$:

   $$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

   Where σ is the standard deviation of the Gaussian kernel.

2. **Gradient Calculation:** The gradient of the smoothed image is calculated using Sobel operators to approximate the derivatives:

   $$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

   where $*$ denotes the convolution operation, and A is the smoothed image. Another operator, like Roberts and Prewitt, can be used instead of Sobel as well.

3. **Edge Magnitude and Direction:** The edge magnitude and direction are determined by:

   $$G = \sqrt{G_x^2 + G_y^2}, \quad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

   The direction is quantified to simplify the edge tracking in the next step.

4. **Non-maximum Suppression:** This step thins the edges by ensuring that only the local maxima are retained, suppressing all other pixels that do not constitute an edge:

   $$\text{If } G \text{ is not a local maximum, set to 0.}$$

5. **Hysteresis Thresholding:** Using two thresholds, weak edges are suppressed unless connected to strong edges:

   $$\text{If } G > \text{high threshold, it's a strong edge;}$$

   $$\text{if } G < \text{low threshold, it's suppressed;}$$

   $$\text{otherwise, it's considered a weak edge.}$$

The information about canny edge detection was sourced from the following [56] [57] [58] [59].

3.2.5 Binarization Techniques

Overview of Binarization

Binarization is the process of converting a grayscale image into a binary image, where the pixels are assigned one of two values, typically 0 and 255. This process simplifies the image, making it easier to analyse and process. [60] [20]

- **Applications**: Binarization is widely used in document image analysis, medical imaging, and various computer vision tasks to facilitate object detection, feature extraction, and image segmentation.[60] [20]

Simple Thresholding (THRESH_BINARY)

THRESH_BINARY is a simple OpenCV thresholding technique where each pixel value is compared to a predefined threshold. If the pixel value is greater than the threshold, it is set to the maximum value (typically 255); otherwise, it is set to the minimum value (typically 0) [60] [20].

- The binary thresholding can be mathematically defined as:

$$I_{binary}(x,y) = \begin{cases} 255 & \text{if } I(x,y) > T \\ 0 & \text{otherwise} \end{cases}$$

  where $I(x,y)$ is the intensity of the pixel at coordinates (x,y) and T is the threshold value.

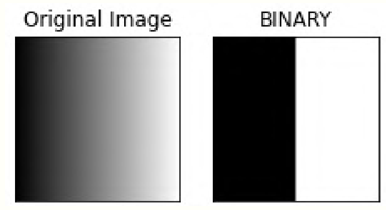


Figure 3.1: Simple Thresholding
source: [60]

- **Applications**: THRESH_BINARY is used in applications where a clear distinction between foreground and background is needed, such as barcode recognition and simple object segmentation [60] [20].

Otsu's Binarization (THRESH_OTSU)

THRESH_OTSU is an adaptive thresholding method in OpenCV that automatically determines the optimal threshold value T from the image histogram. This method aims to minimise the intra-class variance (the variance within the same class) of the black and white pixels [60] [20].

- The Otsu's method determines the threshold by maximising the between-class variance, defined as:
$$\sigma_B^2(T) = w_0(T)w_1(T)[\mu_0(T) - \mu_1(T)]^2$$
where w_0 and w_1 are the probabilities of the two classes separated by threshold T, and μ_0 and μ_1 are the mean intensities of these classes.

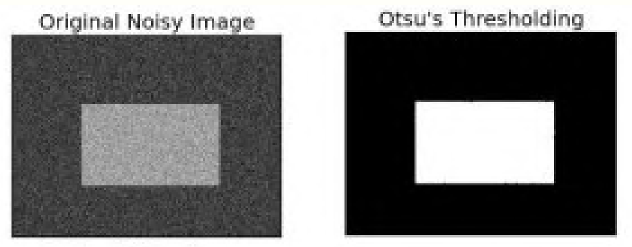


Figure 3.2: Otsu Binarization
source: [60]

- **Applications**: THRESH_OTSU is widely used in image segmentation, particularly for applications requiring flexible thresholding without manual intervention, such as medical imaging, biometric recognition, and automated inspection systems [60] [20] [21].

3.2.6 Morphological Operations

Morphological operations are a set of image processing techniques that process images based on shapes. They apply a structuring element to an input image, creating an output image of the same size.[20]

- **Applications**: These operations are primarily used for preprocessing, such as noise removal, image enhancement, and feature extraction.[20]

Erosion

Erosion is a morphological operation that shrinks the boundaries of the foreground object. It works by moving a structuring element across the image and setting a pixel to the minimum value within the window defined by the structuring element.[61]

- The erosion of image A by structuring element B is defined as:
$$A \ominus B = \{z \in E \mid (B)_z \subseteq A\}$$
where $\ominus$ denotes erosion and $(B)_z$ is the translation of B by z.

- **Applications**: Erosion removes small noise, detaches two connected objects, and shrinks object boundaries.[61]

Dilation

Dilation is the opposite of erosion. It grows the boundaries of the foreground object by moving a structuring element across the image and setting a pixel to the maximum value within the window defined by the structuring element.[61]

- The dilation of image A by structuring element B is defined as:

  $$A \oplus B = \{z \in E \mid (B^*)_z \cap A \neq \emptyset\}$$

  where $\oplus$ denotes dilation and B^* is the reflection of B.

- **Applications**: Dilation is used to fill small holes, connect adjacent objects, and expand object boundaries.[61]

Opening

Opening is a combination of erosion followed by dilation. It is useful for removing small objects from an image while preserving the shape and size of larger objects in the image.[61]

- The opening of image A by structuring element B is defined as:

  $$A \circ B = (A \ominus B) \oplus B$$

  where $\circ$ denotes opening.

- **Applications**: Opening is used to remove noise, separate objects that are close to each other, and smooth the contour of an object.[61]

Closing

Closing is a combination of dilation followed by erosion. It is useful for closing small holes and gaps within objects in an image.[61]

- The closing of image A by structuring element B is defined as:

  $$A \bullet B = (A \oplus B) \ominus B$$

  where $\bullet$ denotes closing.

- **Applications**: Closing is used to fill small holes, connect broken parts of an object, and smooth the contour of an object.[61]

3.2.7 Template Matching

Template matching is a technique in digital image processing that finds parts of an image that match a template image. It involves sliding the template image over the input image (as in a convolution) and measuring the similarity between the template and the overlapping sub-region of the input image.[22]

- **Applications**: This method is widely used in object detection, pattern recognition, and image registration. Typical applications include face recognition, locating specific features in satellite imagery, and identifying defects in industrial inspection.[22]

Template Matching Algorithm

- **Process**: The process involves comparing the template with sub-images of the same size in the target image and using a metric to measure the similarity. [46]
- **Equations**:
  - **Method: TM_SQDIFF** The TM_SQDIFF method computes the sum of squared differences between the template and the image. This method is sensitive to differences in intensity. [46]

    $$R(x,y) = \sum_{x',y'} \left(T(x',y') - I(x+x',y+y')\right)^2$$

    where $R(x,y)$ is the result matrix, T is the template, and I is the input image.
  - **Method: TM_SQDIFF_NORMED** The TM_SQDIFF_NORMED method normalises the result of the sum of squared differences to account for variations in the intensity of the template and the image. [46]

    $$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$
  - **Method: TM_CCORR** The TM_CCORR method calculates the cross-correlation between the template and the image, measuring the similarity between them. [46]

    $$R(x,y) = \sum_{x',y'} \left(T(x',y') \cdot I(x+x',y+y')\right)$$
  - **Method: TM_CCORR_NORMED** The TM_CCORR_NORMED method normalises the cross-correlation result, making it invariant to changes in the template and image intensities. [46]

    $$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

- **Method: TM_CCOEFF** The TM_CCOEFF method computes the correlation coefficient, adjusting for the mean intensity of the template and the image region. [46]

$$R(x,y) = \sum_{x',y'} \left(T'(x',y') \cdot I'(x+x',y+y')\right)$$

  where

$$T'(x',y') = T(x',y') - \frac{1}{w \cdot h} \sum_{x'',y''} T(x'',y'')$$

$$I'(x+x',y+y') = I(x+x',y+y') - \frac{1}{w \cdot h} \sum_{x'',y''} I(x+x'',y+y'')$$

- **Method: TM_CCOEFF_NORMED** The TM_CCOEFF_NORMED method normalises the correlation coefficient, ensuring the result is less affected by intensity variations. [46]

$$R(x,y) = \frac{\sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

- **Advantages**: Template matching is simple and effective for well-defined objects and scenes with minimal background clutter. [46]
- **Limitations**: It is computationally expensive and sensitive to changes in scale, rotation, and illumination. [46]

Template Matching with OpenCV

OpenCV provides efficient and easy-to-use functions for template matching. The library supports various methods, such as squared difference and normalised cross-correlation.[45]

- **Implementation**:

```

# perform template matching
res = cv.matchTemplate(img_gray,template,cv.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where( res >= threshold)
for pt in zip(*loc[::-1]):
 cv.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,0,255), 2)

# Show result
cv2.imshow('Result', img_rgb)
cv2.waitKey(0)

```

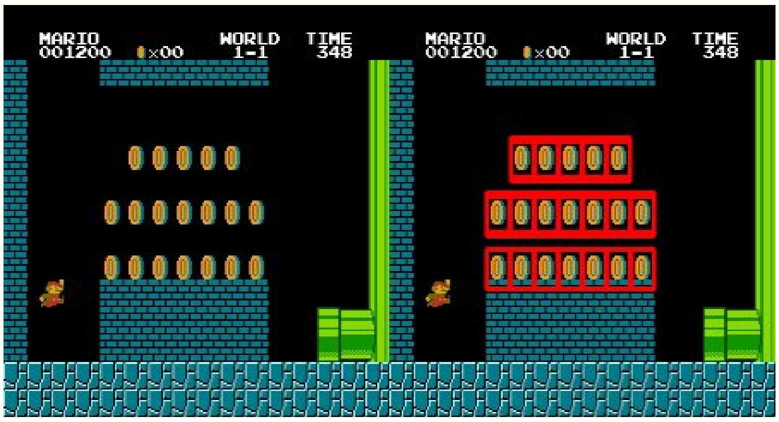


Figure 3.3: Template Matching
source: [45]

3.2.8 SIFT Algorithm

Overview of SIFT

The SIFT algorithm is a computer vision algorithm used to detect and describe local features in images. Developed by David Lowe in 1999, it is widely used for object recognition, image stitching, and 3D reconstruction due to its robustness to changes in scale, rotation, and illumination.[23]

- **Applications**: SIFT is utilised in various fields, including robotics, medical imaging, and augmented reality for tasks such as feature matching, object recognition, and motion tracking [23].

Key Stages of SIFT

- **Scale-Space Extrema Detection**: The first stage involves identifying key points by searching for local extrema in a series of Difference-of-Gaussians (DoG) images, which are obtained by subtracting two consecutive Gaussian-blurred images.[23]

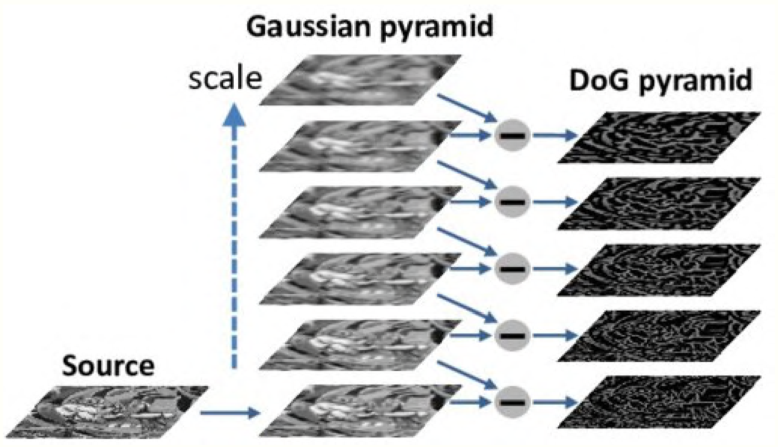


Figure 3.4: SIFT keypoint detection with 6 Gaussian image layers
source: [24]

- **Gaussian Blur:** The image is blurred using a Gaussian function to create progressively smoothed versions of the original image.
- **Difference of Gaussian:** The difference between successive Gaussian-blurred images is calculated to highlight regions of interest (keypoints).

$$D(x,y,\sigma) = (G(x,y,k\sigma) - G(x,y,\sigma)) * I(x,y)$$

where D is the Difference-of-Gaussians image, G is the Gaussian-blurred image at different scales, k is a constant multiplicative factor, and I is the input image.[23]

- **Keypoint Localisation**: Once potential key points are identified, the algorithm refines their positions and scales to ensure stability. This involves fitting a 3D quadratic function to the local sample points to interpolate the location of the maximum or minimum. Therefore, low-contrast points are eliminated, points along edges that might be poorly localised are removed, and the position of each key point is interpolated to improve accuracy. [23]

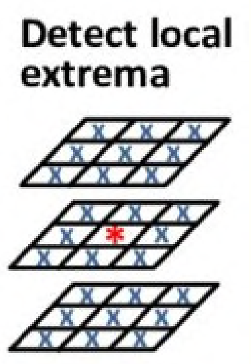


Figure 3.5: Keypoint Localisation
source: [24]

- **Orientation Assignment**: An orientation is assigned to each key point to achieve invariance to image rotation. This is done by computing the gradient magnitude and orientation in the local neighbourhood of the keypoint [23].

$$\theta(x,y) = \tan^{-1}\left(\frac{L_y(x,y)}{L_x(x,y)}\right)$$

where $\theta(x,y)$ is the orientation, L_y and L_x are the image gradients in the y and x directions, respectively.

$$M(x,y) = \sqrt{L_x(x,y)^2 + L_y(x,y)^2}$$

where $M(x,y)$ is the gradient magnitude.

- **Keypoint Descriptor**: A descriptor is computed for each key point by considering the local gradients around the key point. The gradients are weighted by a Gaussian window and then accumulated into orientation histograms, creating a 128-element feature vector for each key point [23].

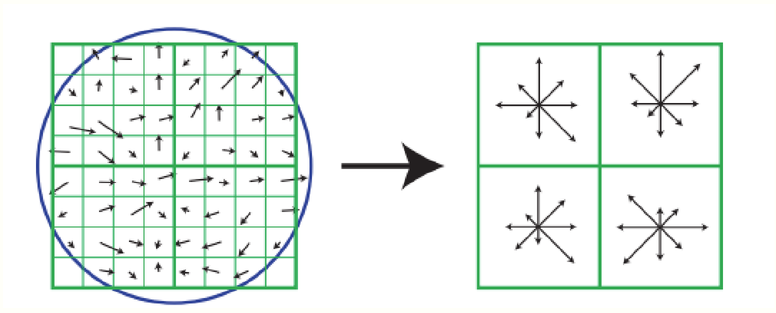


Figure 3.6: Image Gradient and Keypoint Descriptor
source: [23]

Python Implementation using OpenCV

In this section, an example code of the sift algorithm will be implemented using Python and OpenCV. The required libraries are OpenCV, Numpy and matplotlib.

The first step is to import the golden reference component and the example component and convert them into grey-scale images.

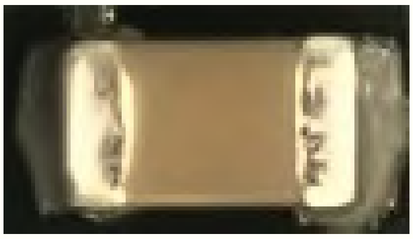


Figure 3.7: Golden Reference Component

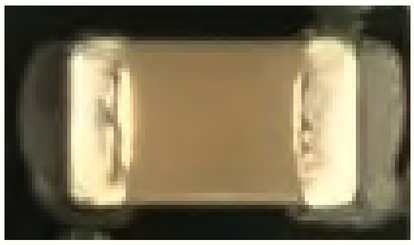


Figure 3.8: example component

After importing the images to the coding environment, the second step is to create a SIFT Detector to detect keypoints.

```

# Create a SIFT detector
sift = cv2.SIFT_create()

# Detect keypoints and compute descriptors
keypoints, descriptors = sift.detectAndCompute(gray_image, None)

# Draw keypoints on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None, flags=cv2.
    DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)


```

A SIFT detector is created to detect key points using OpenCV's `SIFT_create` function. The `detectAndCompute` method detects key points and computes their descriptors, and then the key points are drawn on the Golden reference component for visualisation.

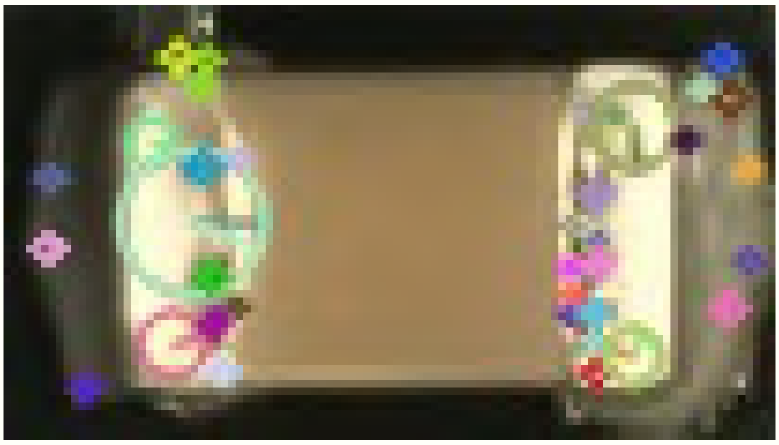


Figure 3.9: Image with Keypoints

Next step is to pass the second image of the example component to the SIFT algorithm to detect keypoints then match it with the original template.

```

# Detect keypoints and compute descriptors for the second image
keypoints2, descriptors2 = sift.detectAndCompute(gray_image2, None)

# Match descriptors using BFMatcher
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
matches = bf.match(descriptors, descriptors2)
matches = sorted(matches, key=lambda x: x.distance)

# Draw matches
matched_image = cv2.drawMatches(image, keypoints, image2, keypoints2,
    matches[:50], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

```

The `BFMatcher` (Brute Force Matcher) is used to match descriptors between the two images.

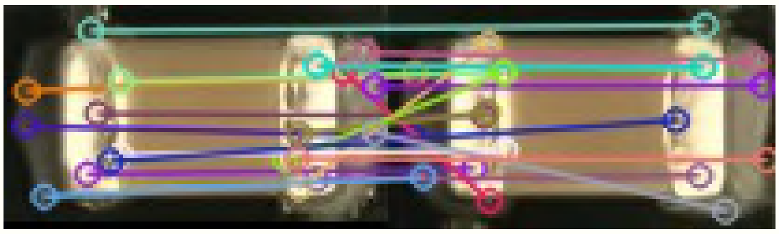


Figure 3.10: Matched Image

The performance of the sift algorithm can then be evaluated by the following code

```

# Evaluate the performance of keypoint matching
```

```python
print(f"Number of keypoints in image 1: {len(keypoints)}")
print(f"Number of keypoints in image 2: {len(keypoints2)}")
print(f"Number of matches: {len(matches)}")

# Calculate matching accuracy
matching_accuracy = len(matches) / min(len(keypoints), len(keypoints2))
print(f"Matching accuracy: {matching_accuracy:.2f}")

```

Example Output:

```
Number of keypoints in image 1: 46
Number of keypoints in image 2: 31
Number of matches: 20
Matching accuracy: 0.65
```

Difference Between BFMatcher and FLANN BasedMatcher

In computer vision, matching feature descriptors between images is a common task. OpenCV provides two popular methods for this purpose: BFMatcher and FLANN BasedMatcher. Both are used to find correspondences between keypoints in different images, but they differ in their approaches, efficiency, and use cases.

- **BFMatcher:** Brute Force Matcher (BFMatcher) is a straightforward and brute-force method for finding matches between descriptors. It compares each descriptor in the first set with each descriptor in the second set and returns the best matches. [62]

  Key Characteristics:

  - **Method:** Compares every descriptor from the first set with every descriptor from the second set. [62]
  - **Distance Measurement:** Supports various distance metrics like L2 (Euclidean distance), L1 (Manhattan distance), and Hamming distance (for binary descriptors). [62]
  - **Simplicity:** Simple to use and implement. [62]
  - **Speed:** Slower for large datasets due to its brute-force nature. [62]

- **FLANN BasedMatcher:**

  Fast Library for Approximate Nearest Neighbors BasedMatcher (FLANN) is designed for large datasets and high-dimensional spaces. It uses efficient approximate nearest-neighbour search algorithms to find matches quickly. [63]

  Key Characteristics:

  - **Method:** Uses approximate algorithms for nearest neighbour search, such as KD-trees and hierarchical clustering. [63]

- **Speed:** Much faster than BFMatcher for large datasets due to the approximate nature of the search.[62]
- **Scalability:** Scales well with the size of the dataset and the dimensionality of the descriptors. [62]
- **Flexibility:** Automatically selects the best algorithm based on the provided parameters. [62]

Table 3.1: Comparison of BFMatcher and FLANN BasedMatcher

Aspect	**BFMatcher**	**FLANN BasedMatcher**
Method	Brute-force comparison	Approximate nearest neighbor search
Speed	Slower for large datasets	Faster, especially for large datasets
Accuracy	Exact matches	Approximate matches
Use Case	Small datasets, high accuracy required	Large datasets, real-time applications
Distance Metrics	L1, L2, Hamming, etc.	Typically L2 for continuous descriptors
Implementation	Simple and easy to use	More complex, requires parameter tuning

BFMatcher and FLANN BasedMatcher serve different purposes based on the size and nature of the dataset. BFMatcher is straightforward and suitable for smaller datasets where accuracy is essential. FLANN BasedMatcher, on the other hand, excels with large datasets, providing faster approximate matches, making it ideal for real-time applications. By understanding the strengths and limitations of each, you can choose the appropriate matcher for your specific computer vision task.

3.2.9 Template Matching and SIFT Algorithm Code in Inspection Process

In this part, we will go over the code implementation done in this thesis for both template matching and sift algorithm.

Template Matching Code Implementation

The following code snippet performs template matching to identify and locate components on a printed circuit board (PCB) using a primary reference image (golden component) and, if necessary, secondary reference images. Below is a detailed explanation of the code.

```
temp_max = -1
# Perform template matching with the golden component reference
res = cv2.matchTemplate(inspect_component, template, cv2.TM_CCOEFF_NORMED)
_, max_val, _, max_loc = cv2.minMaxLoc(res)
threshold = 0.7 # Set threshold at 70%
```

Explanation:

- `temp_max` is initialised to -1 to store the maximum correlation value found during template matching.
- `cv2.matchTemplate` is used to perform template matching between the component to be inspected (`inspect_component`) and the golden reference component (`template`). The `cv2.TM_CCOEFF_NORMED` method is used to normalise the correlation coefficients.
- `cv2.minMaxLoc` is used to find the minimum and maximum correlation values and their locations in the result matrix `res`.
- A threshold of 0.7 is set to determine whether the template matching is successful.

```
# If template matching fails, loop through all secondary component
# reference images and use them as a template.
if max_val < threshold:
    secondary_template_catalogue = self.templateDict.get(name.split("#")[0], pd.DataFrame())
    # Loop through all secondary templates of good components
    for index, row in secondary_template_catalogue.iterrows():
        # extract the good component template image
        # perform template matching using the secondary good component template.
        res = cv2.matchTemplate(inspect_component, secondary_template, cv2.TM_CCOEFF_NORMED)
        _, max_val, _, max_l = cv2.minMaxLoc(res)
        # if template matching is higher than the threshold, replace the old template with the new.
        if max_val >= threshold:
            if temp_max < max_val:
                max_loc = max_l
                temp_max = max_val

if temp_max >= threshold:
    max_val = temp_max
```

Explanation:

- If the initial template matching with the golden component fails (`max_val < threshold`), the code attempts to use secondary reference images stored in `secondary_template_catalogue`.
- The `self.templateDict` dictionary retrieves the secondary templates based on the component name.
- A loop iterates through each secondary template, performing template matching as described above.
- If the correlation value for a secondary template exceeds the threshold and is higher than the previous maximum (`temp_max`), the location and value are updated.
- After checking all secondary templates, if a match exceeding the threshold is found, `max_val` is updated to `temp_max`.

```
# If the primary and secondary good components templates fail to match the
   component
# perform SIFT. Otherwise, the component is found and matched.
if max_val < threshold:
    # use SIFT
else:
    # component found
```

Explanation:

- If neither the primary nor the secondary templates provide a match (`max_val < threshold`), the code proceeds to use the SIFT algorithm for more robust feature matching.
- If a match is found (`max_val >= threshold`), it indicates that the component has been successfully identified and located using template matching.

SIFT Code Implementation

The following code snippet uses the SIFT algorithm to calculate the rotation angle between a test image and a component image. This is achieved by detecting keypoints, computing descriptors, matching these descriptors using the FLANN matcher, and then finding the homography to compute the rotation angle. Below is a detailed explanation of the code.

SIFT initialisation

```python

def matching_and_rotation_calculation(self, test_img, component_img):
    # Initialise SIFT detector
    sift = cv2.SIFT_create()
```

- `sift = cv2.SIFT_create()` initialises the SIFT detector.

Finding Keypoints

```python

    keypoints1, descriptors1 = sift.detectAndCompute(test_img, None)
    keypoints2, descriptors2 = sift.detectAndCompute(component_img, None)

    # Check if descriptors are detected
    if descriptors1 is None or descriptors2 is None:
        return False, 0
```

- `keypoints1, descriptors1 = sift.detectAndCompute(test_img, None)` detects keypoints and computes descriptors for the test image.
- `keypoints2, descriptors2 = sift.detectAndCompute(component_img, None)` detects keypoints and computes descriptors for the component image.

FLANN initialisation

```python

    # Use FLANN for matching
    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params, search_params)
```

- FLANN-based matcher is initialised with `index_params` and `search_params`.

Matching descriptors

```python

    # Adjust k based on the number of descriptors
    if len(descriptors1) < 2 or len(descriptors2) < 2:
        k_dis = 1
    else:
        k_dis = 2

    matches = flann.knnMatch(descriptors1, descriptors2, k=k_dis)
```

- `k_dis` is set to 2 if both sets of descriptors have more than one entry, otherwise it is set to 1.

- matches = flann.knnMatch(descriptors1, descriptors2, k=k_dis) finds the k-nearest matches between descriptors.

Matches Thresholding

```python

# Apply ratio test with different handling based on the value of k_dis
good_matches = []
if k_dis == 2:
    good_matches = [m for m, n in matches if m.distance < 0.85 * n.distance]
else:
    good_matches = [m[0] for m in matches]  # All matches are considered good if k=1
```

- A ratio test is applied to filter good matches:
  - If k_dis == 2, matches are filtered using Lowe's ratio test.
  - If k_dis == 1, all matches are considered good.

Homography Matrix

```python

# Ensure that we have enough good matches to proceed
if len(good_matches) > 5:
    src_pts = np.float32([keypoints1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
    dst_pts = np.float32([keypoints2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)

    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
```

- If there are more than 5 good matches, the source and destination points are extracted.
- M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0) computes the homography matrix.

Finding Rotation

```python

    # Check if the homography matrix is valid
    if M is not None and M.shape == (3, 3) and np.linalg.det(M) != 0:
        theta = -math.atan2(M[0, 1], M[0, 0]) * (180 / math.pi)
        return True, theta
return False, 0
```

- The validity of the homography matrix is checked. If valid, the rotation angle theta is calculated using the arctangent of the matrix elements.
- If all conditions are met, the function returns True and the rotation angle theta. Otherwise, it returns False, 0.

3.2.10 Trilateration

Overview of Trilateration

Trilateration is a mathematical method used to determine the position of a point by measuring its distances to at least three known points. This technique is widely used in geolocation, navigation, and surveying. [64]

- **Applications**: Trilateration is utilised in Global Positioning System (GPS) technology, cellular network positioning, and indoor navigation systems. It helps in accurately determining the location of objects or devices.

Mathematical Basis of Trilateration

2D Trilateration: In two-dimensional trilateration, the position of a point is determined using the distances from three known points. This involves solving for the intersection of three circles; each centred at a known point with a radius equal to the distance to the point being located. [64]

3.3 Tools

A variety of tools were utilised to build and deploy the project effectively. These tools were chosen for their specific functionalities that cater to different aspects of the project, from data handling and image processing to Graphical User Interface (GUI) development and machine learning. Each tool plays a crucial role in the overall workflow, contributing to the robustness and efficiency of the development process. Pickle was used to store data. OpenCV, Python Imaging Library, matplotlib and Scikit-image were used for the powerful image manipulation and processing tools they provide. PyQt was used because it offers powerful tools to build GUIs that can work on different operating systems. The CNN chosen for this project was YOLO Ultralytics, as it provides comprehensive documentation and tools that are necessary for this project. These tools are used as through them, we can apply the methodology previously discussed.

3.3.1 PKL Process

PKL files hold serialised data. Data can be serialised and deserialised for use by different programs, a process known as ”pickling” and ”unpickling”. Pickling is storing a Python variable in a file or a database or transmitting it over a network by converting it into a data form of byte stream. This byte stream can be deserialised into a Python variable by unpickling. PKL files preserve data, making it easier to resume processing later and facilitating the exchange of Python variables between different programs and over the network. However, PKL files might need help with large datasets or real-time data transfer. [65]

3.3.2 The Python Imaging Library

The PIL, or Python Imaging Library, is an extension that enhances the image processing features of the Python programming language. It has extensive support for various file formats and robust image processing and graphic design capabilities [66].

Key Features:

- **File Format Support:** PIL supports a wide variety of image file formats, including PNG, JPEG, GIF, TIFF, and BMP.
- **Image Processing:** The library provides numerous functions for image processing, such as filtering, enhancing, and transforming images.
- **Graphics Capabilities:** PIL allows for basic drawing operations on images, such as drawing text, lines, and shapes.

3.3.3 PyQt

PyQt6 is a Python package that uses a Qt application framework set that enables the creation of sophisticated and highly customisable graphical user interfaces (GUIs). Released in 2021, PyQt6 leverages the powerful features of Qt, providing tools for building cross-platform applications that run seamlessly on Windows, macOS, and Linux. The library offers extensive support for widgets, graphics, and other GUI components, allowing developers to create simple and complex user interfaces easily. [67] PyQt supports various modern GUI functionalities, including advanced 2D and 3D graphics, database integration, and network communication. This makes it suitable for multiple applications, from desktop software to specialised scientific and engineering tools. Furthermore, PyQt6's compatibility with other Python libraries allows for integrating data processing and visualisation capabilities, helping create interactive applications. [67]

3.3.4 matplotlib

Matplotlib, developed by John D. Hunter in 2003, is a widely used plotting library for Python, enabling the creation of static, interactive, and animated visualisations. Matplotlib provides an extensive application programming interface (API) that mimics MATLAB's plotting capabilities, making it familiar to users with a background in MATLAB. The library supports various plot types, including line plots, bar charts, histograms, scatter plots, and more complex visualisations like 3D plots. Matplotlib's integration with NumPy and pandas facilitates the seamless visualisation of data arrays and DataFrame objects, enhancing its utility for scientific research and data analysis. One of Matplotlib's significant strengths is its highly customisable nature, allowing users to fine-tune every aspect of their plots, from colours and fonts to line styles and markers. This flexibility is essential for producing publication-quality figures that meet specific formatting requirements. Additionally, Matplotlib's ability to output in various formats, such as PNG, PDF, SVG, and EPS, makes it a versatile tool for applications from web development to academic publishing. The library's extensive documentation and active community support further contribute to its prominence as a go-to tool for data visualisation in Python [68].

3.3.5 Scikit-image

Scikit-image (skimage) is a Python library for image processing, part of the scikit-learn ecosystem. It offers various algorithms for filtering, segmentation, feature extraction, and transformation tasks. The library is efficient for large-scale image analysis, has a user-friendly interface, and integrates with Python libraries. It is widely used in medical imaging, astronomy, machine learning, and computer vision projects.[69]

3.3.6 Ultralytics YOLO

YOLO is a system that detects objects in real-time. It was created in 2015 by Joseph Redmon and Ali Farhadi. The system was initially developed using the Darknet framework. YOLO uses a single-stage detector and a CNN to directly predict objects' bounding boxes and class probabilities from input images. YOLO has an excellent architecture and algorithm for object detection utilising CNN to achieve a balance between processing speed and accuracy. [70]

It works by dividing the input image into a grid to estimate the possibility of object presence, the object's bounding box, and its class category for each grid cell, all in one step, differing from two-stage detectors like R-CNN, making it faster and more efficient.[70]
The YOLO architecture employs three fundamental concepts for object detection.

- YOLO uses residual blocks to segment the image into grids, which act as reference points, and each has predictions attached to it. [71]
- it then considers each grid's reference points as a base for generating bounding boxes. These bounding boxes are important for identifying object classes. [71]
- Then YOLO uses the intersection method over Union to select the most appropriate bounding boxes. [71]

More in-depth, YOLO simplifies detection by dividing input images into a grid. Each cell in the grid has to predict object presence and their bounding box coordinates. Here is the flow of process

- The algorithm utilises a CNN to process the input image and extract relevant features [70].
- These extracted features are processed by a series of fully connected layers that predict class probabilities and the dimensions of bounding boxes [70].
- Concurrently, the image is divided into a grid, with each grid cell predicting possible bounding boxes and their associated class probabilities [70].
- The output of the network includes bounding boxes and class probabilities for each grid cell [70].
- To enhance the accuracy of predictions, a method called non-max suppression is applied to eliminate redundant, overlapping boxes, keeping only the most likely ones [70].
- The final result of this process is a set of definitive bounding boxes and class identifications for the detected objects [70].

YOLOv8

YOLOv8, developed by Ultralytics, is the latest iteration in the YOLO series of object detection models. It offers significant improvements in accuracy, speed, and versatility for real-time object detection tasks.[72]

YOLOv8 supports the following tasks:

- **Detection:** identifying object class and location [73].
- **Segmentation:** step up from detection involves identifying individual objects and segmenting them [73].
- **Pose Estimation:** involves identifying specific points in an object. Example joints [73].

- **Tracking:** provide a unique ID for each object Identified and track it [73].
- **Classification:** simply take an object and give it a unique ID [73].

These tasks provide a tool that can be embedded in diverse applications. users can easily create their own model using YOLOv8 as a base and feed YOLOv8 their prepared dataset to fine-tune it to detect the desired objects. [72]

YOLOv8 Architecture

YOLOv8 represents an evolutionary step forward in the YOLO series of object detection algorithms, featuring several architectural improvements aimed at enhancing and refining detection accuracy, speed and efficiency:

Backbone

- **CSPDarknet53 Architecture:** The backbone of YOLOv8 is built upon a revised CSPDarknet53 architecture, which comprises 53 convolutional layers. It features cross-stage partial connections that enhance the flow of information across layers, contributing to more effective feature learning.[51]

Neck

- Referred to as the feature extractor, it integrates feature maps from various phases of the backbone to gather information across multiple scales. YOLOv8 introduces an innovative C2f module, which replaces the conventional FPN. This new module merges high-level semantic features with low-level spatial details, enhancing detection precision, particularly for smaller objects.[51]

Head

- **Convolutional Layers:** The head of the network consists of multiple convolutional layers followed by fully connected layers. These are primarily responsible for predicting bounding boxes, objectness scores, and class probabilities.[51]

Enhancements

- **Self-Attention Mechanism:** A new addition to YOLOv8 is integrating a self-attention mechanism within the head. This mechanism allows the model to selectively focus on different parts of the image, adjusting the importance of features based on their relevance to the detection task [51][52][53].
- **Multi-Scale Detection:** YOLOv8 incorporates a feature pyramid network to excel in multi-scale object detection. This capability allows the model to detect objects of various sizes more accurately, from large to small, across different scales of input [51][52][53].

- **Anchor-Free Detection:** The architecture has moved towards an anchor-free detection mechanism that predicts bounding boxes centred on objects directly. This shift eliminates the need for predefined anchor boxes, thus enhancing the model's adaptability and precision in varying object sizes and shapes [51][52][53].
- **optimised Modules and Convolutions:** Continuous refinements include optimised modules and convolutions that boost both the speed and accuracy of the model. These improvements are pivotal for processing large datasets in real-time [51][52][53].
- **Training Enhancements:** Strategic training techniques, such as mosaic augmentation, are employed to prevent overfitting and enhance model accuracy. Additionally, YOLOv8 introduces a decoupled head approach that simplifies the model by removing the objectness branch, which reduces computational demands and accelerates inference times without compromising performance [51][52][53].

YOLOv8-OBB

YOLOv8 introduces an important enhancement with its Oriented Bounding Box (OBB) models. OBB models are made to support object detection for objects positioned at various angles or rotations.[74]
Highlights

- **Oriented Bounding Boxes:** YOLOv8-OBB models are an important advancement in object detection technology, particularly beneficial for detecting objects irrespective of their orientation. This feature is especially valuable in applications like aerial imagery and text detection, where objects may not be aligned horizontally or vertically. [74]
- **Minimised Background Interference:** The ability of OBB models to precisely delineate the bounds of an object significantly reduces background noise. This leads to clearer and more accurate detections, which is crucial for detailed analytical tasks. [74]

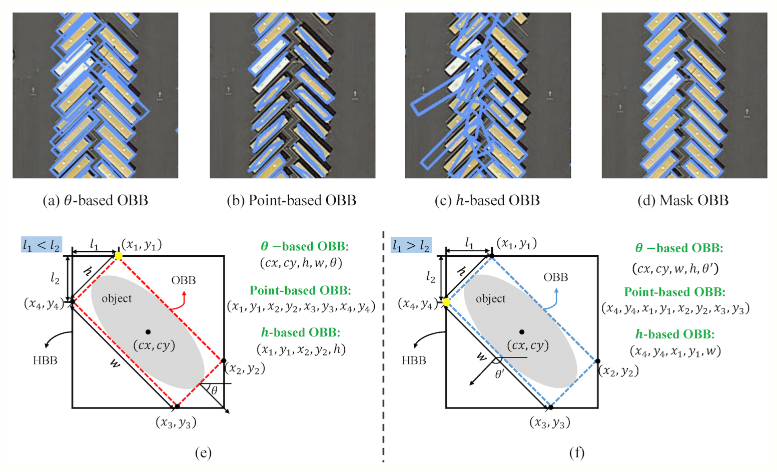
Application of YOLOv8-OBB in this project

- **Component and Defect Detection:** For this thesis, the YOLOv8-OBB model was chosen to detect various components and identify rotation and misalignment defects on PCBs. Components on PCBs may be intentionally or unintentionally rotated during assembly, making traditional bounding boxes less effective. [74]
- **Precision in Detection:** OBB models are good at handling rotations and alignments, which makes them suited for our PCB analysis as some components might be rotated as a defect and cannot be easily found using conventional methods. They ensure that each component's alignment issues are detected, maintaining the high standards required in PCB assembly. [74]

Technical Specifications

- Bounding Box Format: In YOLOv8-OBB, bounding boxes are defined by the coordinates of their four corners, with values normalised between 0 and 1. The typical format used is:

$$\text{class\_index}, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4$$



(a) θ-based OBB
(b) Point-based OBB
(c) h-based OBB
(d) Mask OBB
$l_1 < l_2$
(x_1, y_1)
(x_4, y_4)
object
OBB
HBB
(cx, cy)
(x_2, y_2)
(x_3, y_3)
θ −based OBB: (cx, cy, h, w, θ)
Point-based OBB: $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$
h-based OBB: (x_1, y_1, x_2, y_2, h)
(e)
$l_1 > l_2$
θ −based OBB: (cx, cy, w, h, θ')
Point-based OBB: $(x_4, y_4, x_1, y_1, x_2, y_2, x_3, y_3)$
h-based OBB: (x_4, y_4, x_1, y_1, w)
(f)

Figure 3.11: Example of YOLOv8-OBB
source: [74]

- **Loss Calculation and Output Generation:** Within the YOLO framework, the OBB model adapts the (xywhr) format for computing losses and generating outputs. [74] This format includes:
  - xy: Center coordinates of the bounding box
  - wh: Width and height of the box
  - r: Rotation angle, which denotes the orientation of the box relative to the horizontal

Chapter 4: Preparation and Process

This chapter will cover the following in the following order: a quick summary of the inspection preparation and implementation process, template image board preparation, labelling and Analysis tool for golden component template preparation (GUI + usage), Dataset preparation and AI model training, and finally, the inspection tool for the PCB inspecting process itself (GUI + usage).

4.1 Procedure Overview

- **Image Acquisition:** The first step involves capturing a high-quality image of the new PCB. This image is designated the ”golden reference board image” and is the benchmark for further analysis and comparison. The accuracy and quality of this image are of the utmost importance, as they form the foundation for all subsequent inspection activities.
- **Image Adjustment:** To prepare the golden reference image for detailed component analysis, several adjustments are necessary:
  - **Cropping:** The image is carefully trimmed to eliminate any irrelevant sections, focusing exclusively on the crucial PCB areas for inspection. This step ensures that the analysis tools can operate more efficiently by concentrating on significant elements without distraction.
  - **Rotating:** The orientation of the PCB image is adjusted to conform to standard alignment specifications. This alignment is vital for ensuring that automated tools and algorithms can consistently recognise and analyse components across different images of similar types of PCBs.
- **Integration with Labelling and Analysis Tool:** Following the preparation of the golden reference image, it is processed through a specially developed board labelling and analysis tool. This integration involves several tasks:
  - **Reference Points Definition:** Three specific reference points are established on the PCB. These points anchor the spatial analysis as fixed coordinates for measuring all components’ positions.
  - **Colour Extraction:** The dominant colours of the PCB background are identified. This colour extraction is essential as it helps differentiate components from the background, facilitating more accurate automated analysis.
  - **Component Extraction and Preparation:** This stage involves extraction, examination and preparation of each component identified on the PCB:

    * **Coordinate Identification:** The precise location of each component is determined relative to the established reference points. This localisation is essential for mapping the exact position of components on the PCB.
    * **Image Filtering:** Various filtering techniques are applied to enhance the visibility of components. These include noise filtering, extracting binary representations of components, and edge detection.
  - **Data Storage and Utilisation:** Once all component data have been extracted and catalogued, the prepared board information is stored in a structured format. This repository of the golden board data is essential for supporting the automated inspection of subsequent PCBs of similar types.

- **Implementation in Inspection Processes:** With the golden board's comprehensive preparation and detailed data securely stored, the inspection system is well-prepared to conduct thorough and precise inspections of other PCB images. Referencing the standardised data makes the automated inspection process more efficient and accurate. There are three main types of inspection in this project:
  - The traditional inspection method
  - The AI YOLO inspection method
  - A method that utilises both Traditional and YOLO AI methods

4.2 Golden Board Preparation

The preliminary stage in a PCB inspection involves careful preparation to ensure that the PCB is integrated seamlessly into the inspection pipeline. This initial phase is critical as it standardises the conditions under which the PCB is analysed, ensuring consistency and reliability in subsequent inspection procedures.

Stringent criteria must be adhered to when documenting the initial image of the golden board for visual inspection processes. This ensures the consistency and reliability of the visual inspection system. These criteria are critical in optimising the processing time and minimising errors during subsequent inspections.

4.2.1 Essential Criteria and Procedures for Image Capture of the Golden Board:

The following part will discuss this thesis's essential criteria for capturing PCB images.

- **Consistent Orientation:** The board's orientation within each image must remain constant across all captures of similar PCB types. This uniformity is crucial as it reduces the need for additional processing to re-align the images during inspection, thereby saving valuable time.

- **Visibility and Placement of Reference Points:** Reference points, essential for the board's calibration and analysis, must be visible and located within the boundaries of the image. Furthermore, these reference points should maintain visibility across all subsequent board images to ensure consistent and accurate measurements.
- **Camera Positioning:** The camera's height relative to the board must remain fixed to guarantee that the other images are captured from a consistent perspective. This uniformity helps reduce variations between images, thus facilitating smoother and faster processing.
- **Image Scale Consistency:** The portion of the board's image should remain consistent across images of the same type of board. Maintaining this consistency helps reduce processing time, as the system can reliably anticipate the board's scale in the images.
- **Controlled Lighting Conditions:** To ensure the images are clear and comparable, the lighting conditions under which the board images are captured should be as consistent as possible. Uniform lighting conditions prevent variations in appearance that could lead to inaccuracies in component detection and classification.

4.2.2 Initial Processing Steps for the Golden Board Image:

Upon capturing the first image of the board, which is typically a defect-free version with all components correctly placed, the following steps are undertaken to prepare the image for detailed analysis:

- **Image Rotation:** The board's image is adjusted to the most suitable angle for inspection. All the following boards should also maintain this rotation value for inspection.
- **Image Cropping:** The image is then cropped to reduce excess background and focus closely on the board. This step ensures that the inspection algorithms can concentrate on relevant data without distractions.

Advancement to Component Extraction and Board Information Processing:

Once the image is satisfactorily adjusted, it is advanced to the next stage, where detailed component and board information is extracted. This stage begins the in-depth analysis, where each component is identified, catalogued, and prepared for inspection against the golden reference. This rigorous and detailed approach to the golden board's initial image capture and preparation is fundamental to setting a solid foundation for the automated visual inspection system. Standardising makes the inspection process more efficient, reliable, and capable of accurately detecting and classifying defects on PCBs.

4.3 Board Labelling and Analysis Tool

Following the preparation of the golden board reference image, the next step is to utilise this image within the specialised PCB analyser and Labeller software. This application is a crucial tool for further inspection and analysis of the PCB, which uses methods of trilateration, filtering, edge extraction, binarization, and morphological operations. In addition, it uses PyQt to build the user interface and a pkl file to save the data structure.

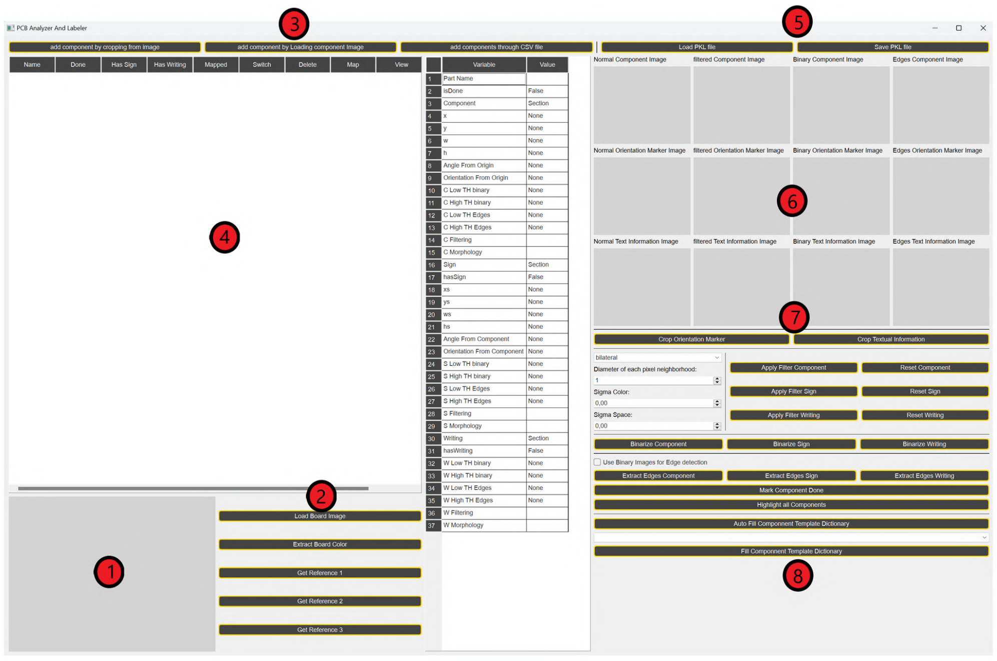
4.3.1 General Overview

As mentioned above, the primary purpose of this tool is to prepare a template of the board to be used in the inspection process. The first step involves loading the golden board image. Once the image is loaded, the program initialises the data structure for this board in the backend. Next, three reference points must be extracted from the image, and the PCB background colour must be extracted. Following this, the components required for inspection are extracted and processed as necessary, which may include operations such as filtering and edge extraction. After all components have been extracted and processed, the data structure is saved as a PKL file. This file can be reloaded into the application later if additional components need editing or extraction.

After preparing the board, you can view all the components together or separately.

4.3.2 Overview of the PCB Analyser and Labeller User Interface

The user interface is built using PyQt. Each element is linked to a function or a process on the backend using PyQt and Python.



	Variable	Value
1	Part Name	
2	isDone	False
3	Component	Section
4	x	None
5	y	None
6	w	None
7	h	None
8	Angle From Origin	None
9	Orientation From Origin	None
10	C Low TH binary	None
11	C High TH binary	None
12	C Low TH Edges	None
13	C High TH Edges	None
14	C Filtering	
15	C Morphology	
16	Sign	Section
17	hasSign	False
18	xs	None
19	ys	None
20	ws	None
21	hs	None
22	Angle From Component	None
23	Orientation From Component	None
24	S Low TH binary	None
25	S High TH binary	None
26	S Low TH Edges	None
27	S High TH Edges	None
28	S Filtering	
29	S Morphology	
30	Writing	Section
31	hasWriting	False
32	W Low TH binary	None
33	W High TH binary	None
34	W Low TH Edges	None
35	W High TH Edges	None
36	W Filtering	
37	W Morphology	

bilateral

Diameter of each pixel neighborhood: 1

Sigma Color: 0,00

Sigma Space: 0,00

Figure 4.1: PCB Analyser and Labeller layout

The user interface of the PCB analyser and Labeller is designed to facilitate an efficient workflow for inspecting and annotating PCBs. Here are the key components of the interface and their functionalities:

1. **Display Area for the Golden Board**:
   - This section prominently displays the golden board image, which serves as the reference for all subsequent analyses.
2. **Control Panel for Board Management**:
   - Users can load the golden board image for the initial setup.
   - The interface for extracting the PCB background colour and defining three reference points directly on the board image is available here.
3. **Component Extraction Controls**:
   - This area provides tools for extracting components from the board using various methods.
4. **Component List Table**:
   - Displays a list of all components identified on the PCB. This table is interactive, allowing users to select and inspect individual components.
5. **File Management**:
   - Located above the component list table, this section includes options to load the board and save the entire board setup as a `.pkl` file.
6. **Component Details and Image Analysis**:
   - Displays images of the selected component, including orientation markers and textual information.
   - Also shows binary and edge detection images.
7. **Toolset for Image Manipulation and Data Extraction**:
   - Offers a variety of tools to extract additional data from components and perform various manipulations to aid in the thorough analysis of PCBs.
8. **Component Library Management**:
   - There is an interface that allows adding similar component types in a single data frame, either from the same image or different images. This is used in inspection when the golden reference fails to identify the inspected component.

4.3.3 PCB analyser Utilisation: Board and Component Preparation

The procedure for preparing a PCB for inspection is methodical and essential for ensuring accurate analysis and quality control. The following steps outline the standardized process utilised within our PCB analyser and Labeller software for setting up a golden board image:

Loading the Golden Board Image:

- Initially, users must load the board image into the system using the designated ”Load Board Image” button located within the control panel for board management, as illustrated below:

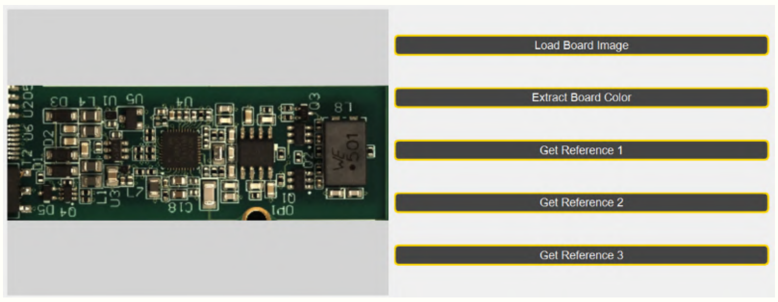


Figure 4.2: Board management interface

- This action initiates the process by uploading the golden board reference image into the tool, setting the stage for subsequent operations.

Color Extraction of the PCB:

- Following the image upload, the next step involves extracting the predominant colour of the PCB. This is accomplished by activating the ”Extract Board Color” button, which triggers the launch of an external application designed for detailed image analysis.

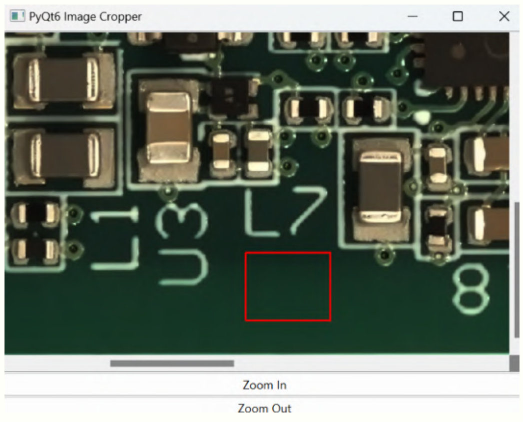


Figure 4.3: Extracting PCB colour

The auxiliary application, Image Cropper, is a versatile tool used across various functionalities within the software. It offers features such as image cropping, zooming, and scrolling for enhanced navigation and precise selection of areas within the image. Once the desired segment of the image is selected, it is returned to the main application for further processing.

1. **Handling Colour Data:**

    - Upon retrieving the cropped image segment, the colour data is processed internally through the `handleExtractColor` function within the main application. This function analyses the image to ascertain each colour channel's minimum and maximum values in the Blue Green Red (BGR), Hue Saturation Value (HSV), and LAB colour spaces.

    ```
            {
            'BGR': {'max': [40, 58, 23], 'min': [32, 46, 18]},
            'HSV': {'max': [78, 170, 58], 'min': [72, 143, 46]},
            'LAB': {'max': [54, 114, 137], 'min': [42, 109, 132]}
            }

    ```

    example of PCB background colour ranges across BGR, HSV, and LAB colour spaces

    - The primary objective of extracting and analysing these colour parameters is effectively distinguishing between the PCB's background and its components. Although the board and components' colours may occasionally be similar, typically, there is a difference that is important for the subsequent inspection processes. Extracting the background colour is a fundamental step in preparing the board for detailed component analysis and quality assessment.

Defining New Origins for Component Localisation:

The process of defining new origins on a PCB involves selecting three reference points essential for the accurate trilateration of components. This methodology is critical for ensuring that each component on the PCB can be precisely located and analysed regardless of variations in image orientation.

Process for Extracting Reference Points:

1. **Selection of Reference Points:**

    - To initiate the localisation process, three reference points on the PCB must be selected. These points serve as anchors for the trilateration process, which is vital for mapping the components accurately during the inspection. The selection is facilitated through this user interface section, which has designated controls for each reference point, as depicted in the figure below.

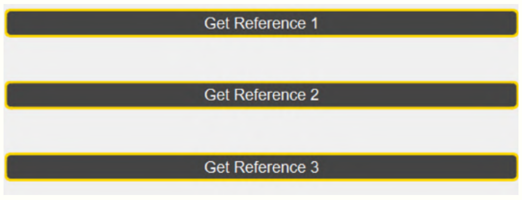


Figure 4.4: Interface for extracting three reference points

- It is imperative to Establishing the first reference point (Reference 1) is imperative before defining the others. This initial point activates the controls for subsequent references, ensuring a systematic approach to the selection process.

2. **Criteria for Reference Point Selection:**

   - The choice of reference points is governed by their visibility and consistency across all boards of the same type. An ideal reference point is typically a circular marker, universally recognised within the PCB industry as a standard for board markers or test points. These markers are consistently positioned and easily recognisable, as shown in the figure below.

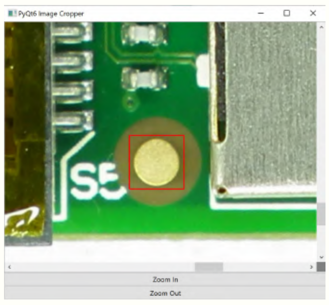


Figure 4.5: Example of a reference point

   - Selecting a circular reference point is advantageous because it remains identifiable under various conditions, including rotations. This attribute is essential for maintaining accuracy in subsequent image-processing steps, especially when board image realignment is required.

3. **Extraction and Handling of Reference Points:**

   - The Image Cropper application is employed once more to facilitate the precise cropping of the selected reference points.
   - Once a reference point is cropped, it is processed internally through the `handleNullPointExtraction` function within the main application. This function captures and records the detailed characteristics of the reference point, ensuring that it can be used effectively in the orientation correction and component localisation processes.

The PCB analyser and Labeller tool establish a foundation for accurate and efficient inspection of assembled circuit boards by defining and extracting these reference points. This process not only enhances the precision of component localisation but also significantly streamlines subsequent inspection routines by stabilising the reference framework against which all components are evaluated.

4. **the role of the reference points:**

   - **Rotation and Translation Adjustment:** They enable the correct orientation and positioning of the board image to match the orientation of the golden template. This adjustment is vital when a new board is loaded for inspection.
   - **Component Localisation:** During inspection, these reference points are used to triangulate the precise location of each component on the board, ensuring that all parts are accurately located when inspected.

Component extraction

Following the successful extraction of the three reference points and the determination of the board's colour, the process transitions to the extraction and processing of components from the board. This phase is crucial for preparing the components for detailed analysis, which includes filtering and binarization to discern fine details such as edges.

Methods for Component Extraction from the Board

Three methods facilitate the extraction of components from the board, each suited to different scenarios and offering varying degrees of control and efficiency

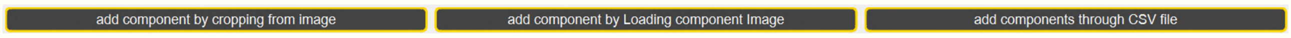

add component by cropping from image
add component by Loading component Image
add components through CSV file

Figure 4.6: Interface for component extraction

1. **Method 1: Direct Cropping**

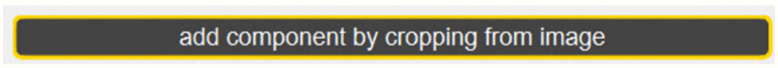


Figure 4.7: Button to extract components from the board image

   - Components are directly cropped from the golden board image using the previously mentioned image cropper application. This process involves extracting the coordinates `(x, y)` along with the width `(w)` and height `(h)` of each component. The coordinates `(x, y)` signify the centre of the component relative to the designated reference point 1 on the board. This method is primarily used for precise, manual extraction of individual components.

2. **Method 2: Adding via Separate Images**

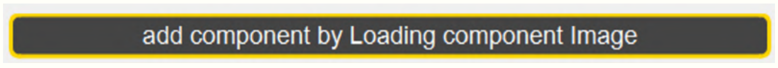
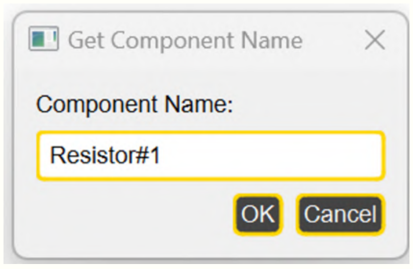


Figure 4.8: Button to add component images from the local storage

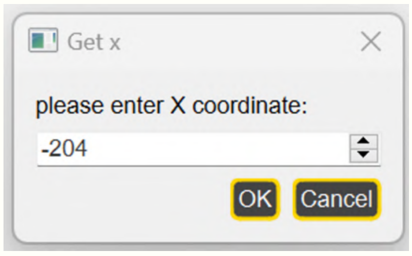
- This approach allows for the incorporation of components using separate images. Each image must include pre-defined dimensions `(w, h)` and centre coordinates `(x, y)`, all referenced to reference point 1 on the board. Users are prompted to input details such as component name, centre coordinates, dimensions, and orientation angle relative to the first reference point. This method is particularly useful for adding components stored as distinct image files, providing flexibility in handling various component images.



Get Component Name

Component Name: Resistor#1

Figure 4.9: Component name input



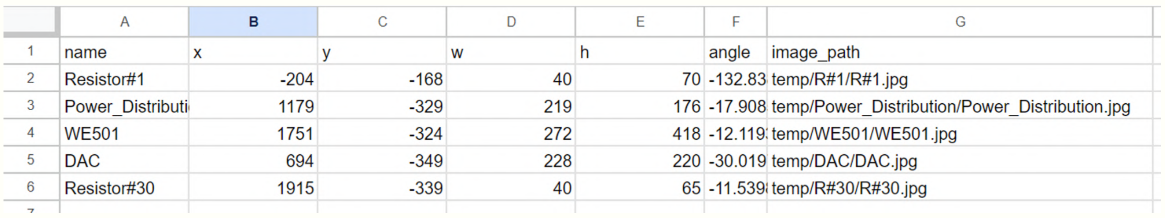
Get x

please enter X coordinate: -204

Figure 4.10: Component coordinate input

3. **Method 3: CSV File Import**

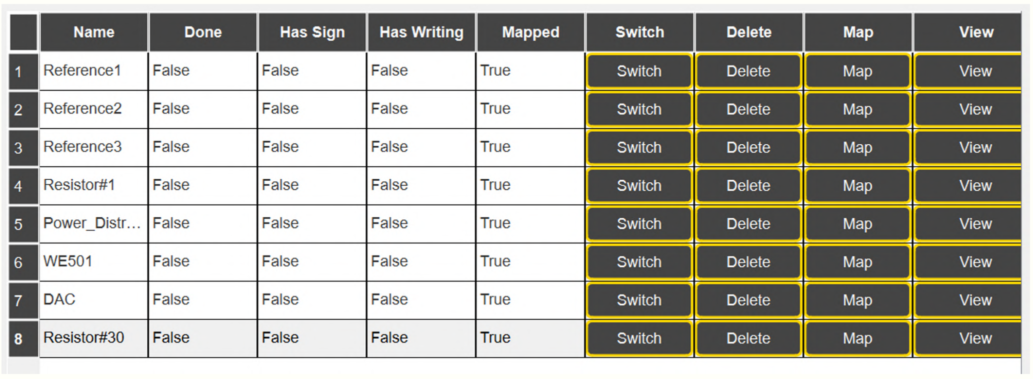
- For bulk processing, components can be imported through a Comma Separated Values (CSV) file that includes information such as the component's name, centre coordinates `(x, y)`, dimensions `(w, h)`, rotation angle, and the path to the component's image.



	A	B	C	D	E	F	G
1	name	x	y	w	h	angle	image_path
2	Resistor#1	-204	-168	40	70	-132.83	temp/R#1/R#1.jpg
3	Power_Distributi	1179	-329	219	176	-17.908	temp/Power_Distribution/Power_Distribution.jpg
4	WE501	1751	-324	272	418	-12.119	temp/WE501/WE501.jpg
5	DAC	694	-349	228	220	-30.019	temp/DAC/DAC.jpg
6	Resistor#30	1915	-339	40	65	-11.539	temp/R#30/R#30.jpg

Figure 4.11: CSV file example with columns name, x, y, w, h, angle and image path

This method is designed to automate the component image-loading process, which is particularly effective when handling a long list of components. The CSV format should be carefully prepared to ensure accurate data entry, as illustrated below. Utilising CSV for component import significantly enhances efficiency, allowing for the mass importation of component data into the application. This is particularly advantageous in production environments where large numbers of components need to be processed swiftly and accurately.



	Name	Done	Has Sign	Has Writing	Mapped	Switch	Delete	Map	View
1	Reference1	False	False	False	True	Switch	Delete	Map	View
2	Reference2	False	False	False	True	Switch	Delete	Map	View
3	Reference3	False	False	False	True	Switch	Delete	Map	View
4	Resistor#1	False	False	False	True	Switch	Delete	Map	View
5	Power_Distr...	False	False	False	True	Switch	Delete	Map	View
6	WE501	False	False	False	True	Switch	Delete	Map	View
7	DAC	False	False	False	True	Switch	Delete	Map	View
8	Resistor#30	False	False	False	True	Switch	Delete	Map	View

Figure 4.12: Filled component table

Viewing and Processing Components

Upon adding components to the inspection system, various functionalities are provided to manage and interact with these components. This section elaborates on the mechanisms available for viewing, deleting, or further processing the components involved in the inspection process.

Interactive Component Visualisation:

1. **Individual Component Viewing:**

   - Each component listed in the system can be viewed individually by pressing the 'View' button associated with the respective component in the table.

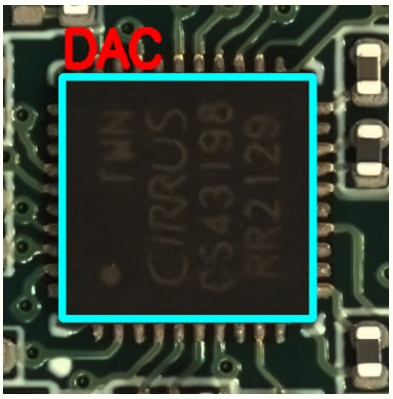


Figure 4.13: Example of labelled component

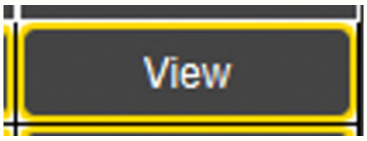


Figure 4.14: Button in the component table to view component

2. **Viewing all components**
   A feature allows users to visualise all components simultaneously by pressing the **Highlight All Components** button, providing a comprehensive view of the board's layout and components.

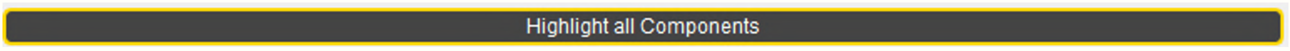


Figure 4.15: Button to view all highlighted components

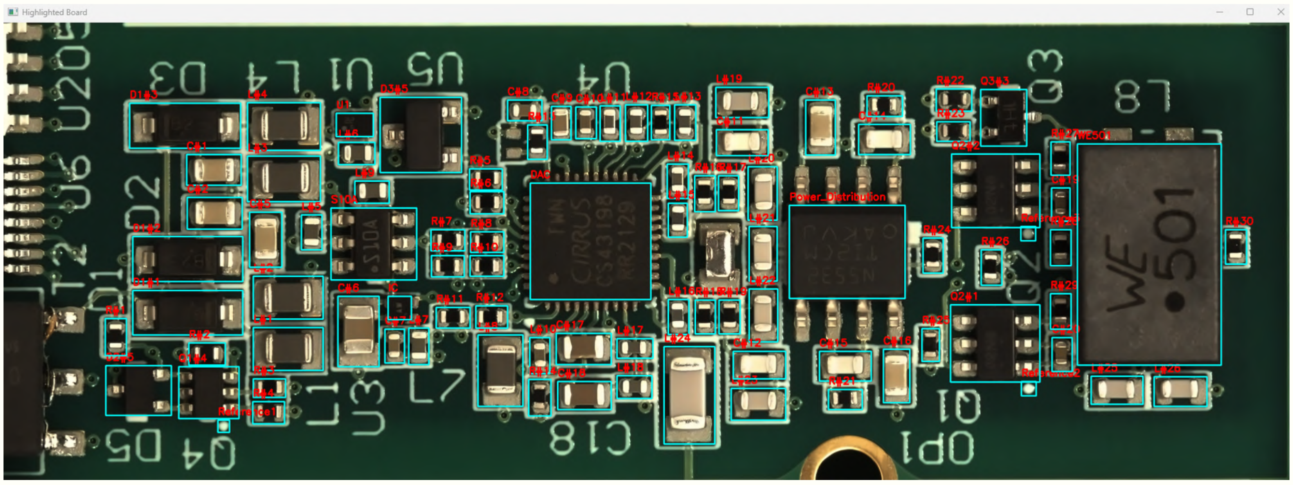


Figure 4.16: Image of highlighted components

3. **Component Deletion**

   - The interface includes an option to remove components by pressing the 'Delete' button located in the component row listing within the components table. This feature ensures that users can easily remove any unwanted or erroneously added components.

4. **Component Selection for Processing**

   - To initiate processing on a specific component, users must first select the component by toggling the 'Switch' button corresponding to the desired component in the components table. This action highlights the component details stored.

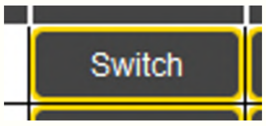
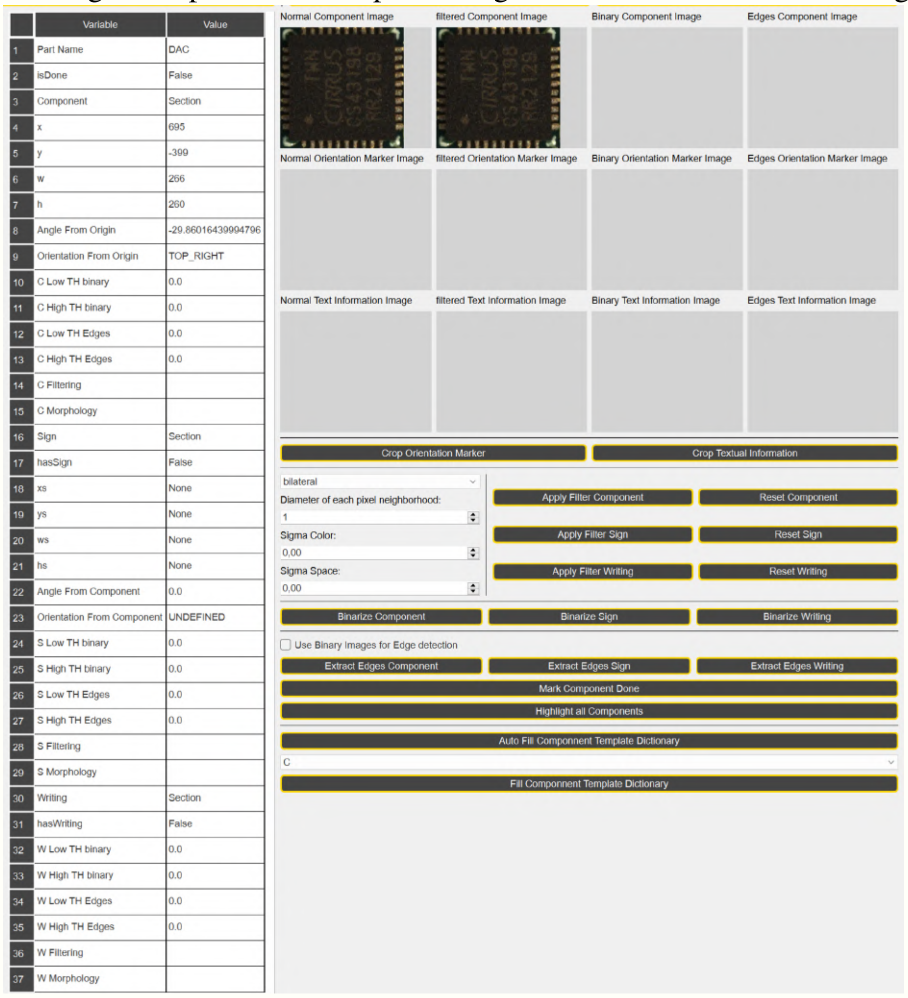


Figure 4.17: Button to switch components in the component table

when selecting a component before processing it, it will be as in the following figure:



	Variable	Value
1	Part Name	DAC
2	isDone	False
3	Component	Section
4	x	695
5	y	-399
6	w	266
7	h	260
8	Angle From Origin	-29.86016439994796
9	Orientation From Origin	TOP_RIGHT
10	C Low TH binary	0.0
11	C High TH binary	0.0
12	C Low TH Edges	0.0
13	C High TH Edges	0.0
14	C Filtering	
15	C Morphology	
16	Sign	Section
17	hasSign	False
18	xs	None
19	ys	None
20	ws	None
21	hs	None
22	Angle From Component	0.0
23	Orientation From Component	UNDEFINED
24	S Low TH binary	0.0
25	S High TH binary	0.0
26	S Low TH Edges	0.0
27	S High TH Edges	0.0
28	S Filtering	
29	S Morphology	
30	Writing	Section
31	hasWriting	False
32	W Low TH binary	0.0
33	W High TH binary	0.0
34	W Low TH Edges	0.0
35	W High TH Edges	0.0
36	W Filtering	
37	W Morphology	

bilateral
Diameter of each pixel neighborhood: 1
Sigma Color: 0,00
Sigma Space: 0,00
Use Binary Images for Edge detection
C

Figure 4.18: Pre-processing component

Summary of the Processing Steps Following the selection of a component, the software provides a structured approach to preparing the component for inspection:

- **Extraction of Orientation Markers and Textual Information:**
  - **Orientation Markers:** These are typically small circular signs on the component that indicate the location of pin 1. Not all components have orientation markers, but when present, they are used for correct component placement and orientation. **Textual Information:** This often includes details such as component type or version. This information is used to verify component specifications and ensure compatibility with the PCB design.
- **Noise Reduction via Filtering:**
  - Applying various filters to the component images helps reduce background noise and enhance the clarity and quality of the visual data required for accurate inspection.
- **Binary Image Extraction and Morphological Operations:**
  - Otsu's thresholding method is utilised to convert images into binary format, followed by morphological operations to refine the image and remove residual noise.

- **Edge Detection:**
  - The final preparatory step involves employing the Canny algorithm to extract the component's edges. Depending on the inspection's specific requirements, this can be performed on either the binary image or the gray-filtered image.
- **Data Preservation:** After processing, all component data and board information can be saved in a `.pkl` file format. This preserves the processed data for future reference or further analysis.

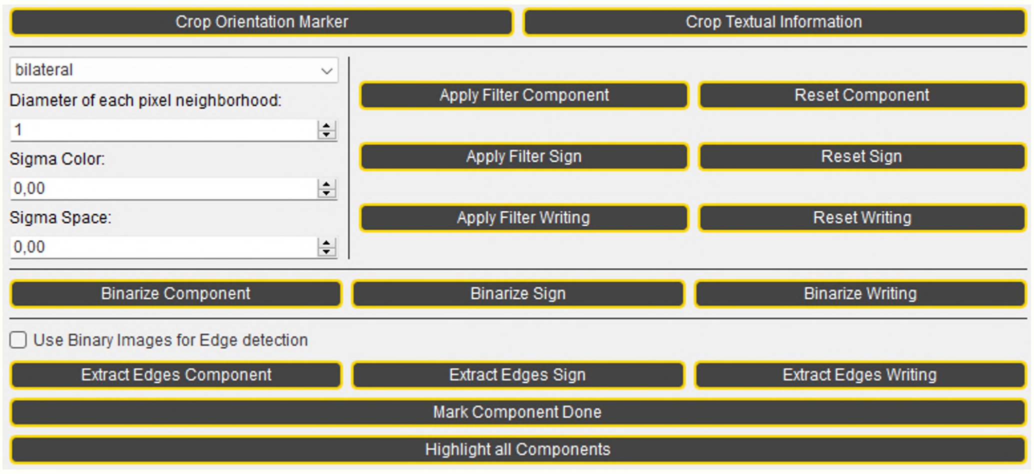


Figure 4.19: Interface for filtering and processing image

- **Filling Component Catalogue:**
  during the inspection process, the golden reference might fail to identify a component, resulting in a False False result. When this happens, the image of the inspected board can be uploaded in the labelled application, and the component image can be added to the component catalogue, which is organised by the type of component, not by the component's position.

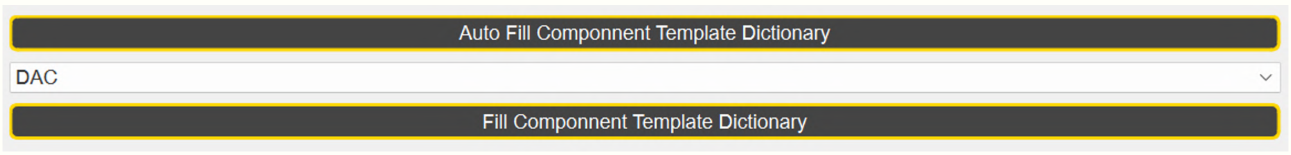

Auto Fill Componnent Template Dictionary
DAC
Fill Componnent Template Dictionary

Figure 4.20: Interface to fill the component catalogue

after filling the catalogue and starting the inspection process, in case the inspection process fails to identify a component, the inspection process goes to the catalogue, extracts the expected component's data frame, and loops through the data frame, comparing the templates saved with the inspected component. Enhancing automation of the inspection process.

4.3.4 Detailed Component Preparation Process

step by step, the following section will show how to process the component and, in the end, view how the component and its information table will be filled.

Extraction of Orientation Markers and Textual Information

Extracting orientation markers and textual information from PCBs is facilitated through the previously mentioned external application known as the Image Cropper. The cropper application interacts internally with the main software via the `handleCroppedImage` function. This function is adept at managing different types of image data depending on the specific user interactions within the graphical interface.

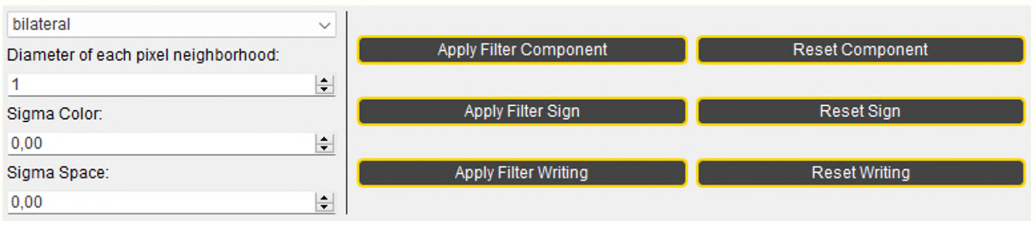
Figure 4.21: Interface to initiate cropping of the Orientation Marker and Textual Information

Upon activation through a user's selection in the aforementioned interface, this function is responsible for processing the image and coordinates extracted by the Image Cropper. Depending on which button the user presses, the function updates the component data with either the orientation markers or textual information linked to specific PCB components.

For instance, when an orientation marker is selected and cropped within the interface, the resultant coordinates are calculated to capture the centre of the orientation marker relative to the centre of the associated component. Additionally, this process involves capturing the angular orientation and the relative angle to the component centre, which is later used to ensure that the component is aligned correctly.

Noise Reduction via Filtering

The subsequent phase in component preparation involves applying filtering techniques to enhance the image quality of the components on the PCBs. This step is vital for optimising the images for further processes, such as binarization and edge detection, which can be part of the detailed inspection and analysis of the components.



bilateral
Diameter of each pixel neighborhood:
1
Sigma Color:
0,00
Sigma Space:
0,00

Figure 4.22: Interface for component image filtering

Within the graphical user interface, users have the ability to select from a variety of filtering techniques via a dropdown menu. Each filtering option is designed to address specific noise issues while preserving essential details of the component imagery:

- **Bilateral Filtering:** is an excellent method for reducing noise while keeping edges sharp, and it is ideal for images where maintaining edge integrity is important.
- **Gaussian Filtering:** A filtering technique that smooths out image noise less aggressively than, suitable for less detailed components.
- **FastNLMeans Denoising:** This method utilises a non-local means denoising algorithm, which is highly effective against random noise. This method excels in retaining detailed textures within the images, thereby enhancing the overall clarity without significant blurring. However, it takes more time to process an image.

An important feature of these filtering operations is their stackability, allowing multiple filtering operations on the same image to refine its quality progressively. Each parameter and operation detail is recorded and stored. This stored data is extracted during the inspection phase, as it allows identical filtering parameters to be applied to the components under inspection. That is why it is important to ensure the consistency of the environment where the board image is captured.

If the filtered results are unsatisfactory, users can easily reverse the changes by pressing the reset button provided within the interface.

Binary Image Extraction and Morphological Operations

Following the initial filtering stage, the process progresses to the binary conversion of the image, which is then subjected to morphological operations to refine the component's visual characteristics.

To initiate this phase, an external application is used for this part, accessible through buttons depicted here:

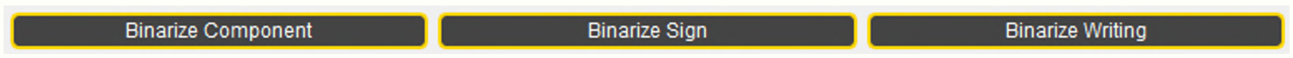


Figure 4.23: Interface to initiate binarization and morphological.

Within this application, the chosen image is then converted into a binary format utilising Otsu's method. After this binary transformation, the user can apply morphological operations such as opening and closing.

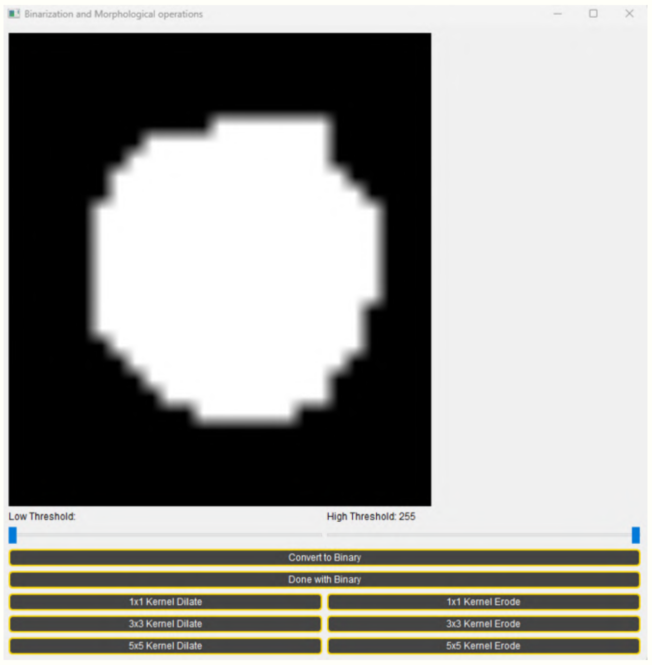


Figure 4.24: The external application interface of binarization and morphological operations.

Note that the slider exists in case another type of binarization is used.

Like the filtering operation, each step within this application is stack-able, allowing for multiple adjustments until the optimal image quality is achieved. Notably, every operation and its parameters are recorded and stored to be then used to process the same component during the inspection phase correctly.

Edge Detection

for the final step, the culmination of the image processing sequence involves the extraction of edges from either the filtered or binary images of the component

To initiate edge detection, you can press one of the following buttons depicted in the figure:

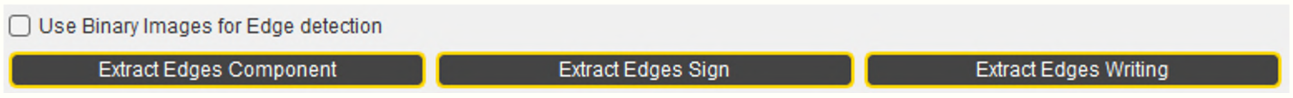
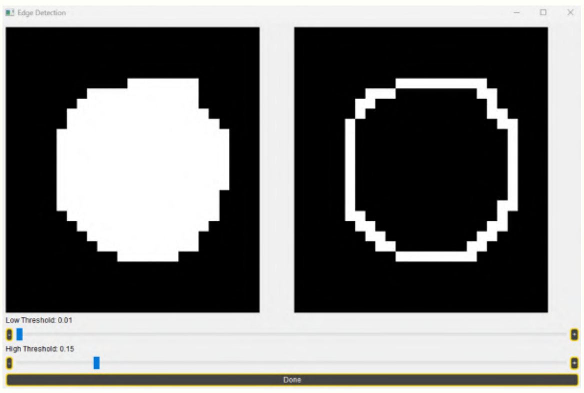


Figure 4.25: Interface to initiate edge extraction.

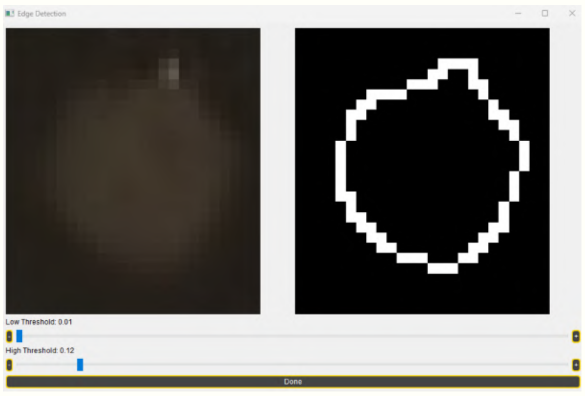
Pressing any of these buttons will call an external application that, through it, you can control the parameter and extract the edges you feel most suitable for the component.



Low Threshold: 0.01

High Threshold: 0.15

Figure 4.26: Edge extraction through binary image



Low Threshold: 0.01

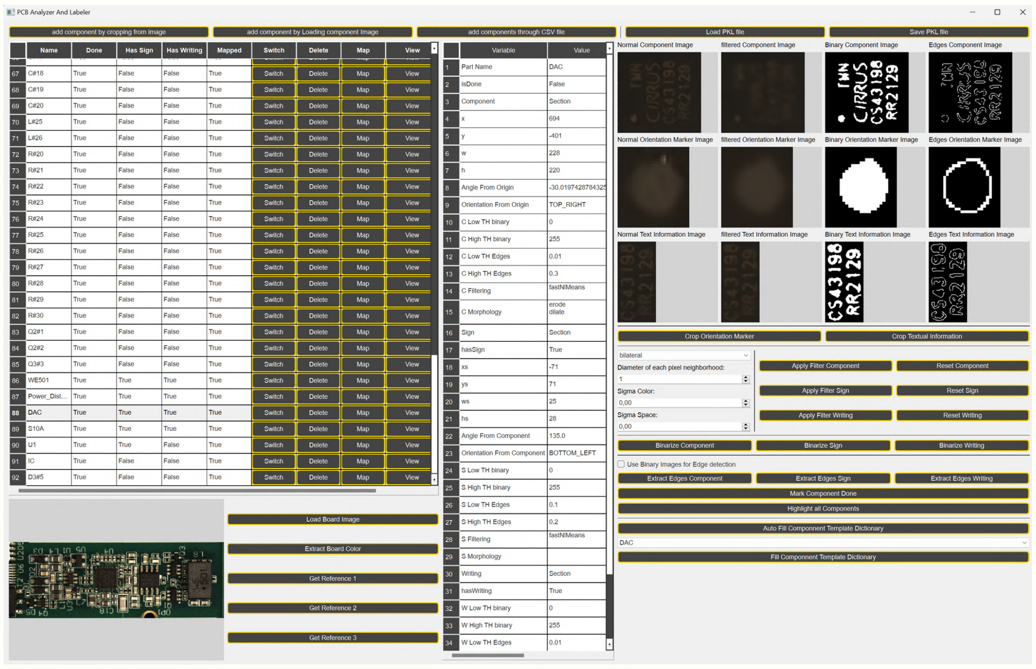
High Threshold: 0.12

Figure 4.27: Edge extraction through grey image

In the examples illustrated above, the application extracts the orientation marker edges once using a binary image and once using a filtered grey image. Similar to the other processing features, the parameter information is stored to process the component to be inspected with the parameters used for the golden component reference.

4.3.5 Data Preservation and Data Structure

Upon completion of labelling and processing all the components and satisfactory results, the application's user interface typically presents a comprehensive overview of the processed components, as illustrated in the following figure. The interface displays all components in a table, where you can individually view, switch, or delete them.



	Name	Done	Has Sign	Has Writing	Mapped	Switch	Delete	Map	View
67	C#18	True	False	False	True	Switch	Delete	Map	View
68	C#19	True	False	False	True	Switch	Delete	Map	View
69	C#20	True	False	False	True	Switch	Delete	Map	View
70	L#25	True	False	False	True	Switch	Delete	Map	View
71	L#26	True	False	False	True	Switch	Delete	Map	View
72	R#20	True	False	False	True	Switch	Delete	Map	View
73	R#21	True	False	False	True	Switch	Delete	Map	View
74	R#22	True	False	False	True	Switch	Delete	Map	View
75	R#23	True	False	False	True	Switch	Delete	Map	View
76	R#24	True	False	False	True	Switch	Delete	Map	View
77	R#25	True	False	False	True	Switch	Delete	Map	View
78	R#26	True	False	False	True	Switch	Delete	Map	View
79	R#27	True	False	False	True	Switch	Delete	Map	View
80	R#28	True	False	False	True	Switch	Delete	Map	View
81	R#29	True	False	False	True	Switch	Delete	Map	View
82	R#30	True	False	False	True	Switch	Delete	Map	View
83	Q2#1	True	False	False	True	Switch	Delete	Map	View
84	Q2#2	True	False	False	True	Switch	Delete	Map	View
85	Q3#3	True	False	False	True	Switch	Delete	Map	View
86	WE501	True	True	True	True	Switch	Delete	Map	View
87	Power_Dist...	True	True	True	True	Switch	Delete	Map	View
88	DAC	True	True	True	True	Switch	Delete	Map	View
89	S10A	True	True	True	True	Switch	Delete	Map	View
90	U1	True	True	False	True	Switch	Delete	Map	View
91	IC	True	False	False	True	Switch	Delete	Map	View
92	D3#5	True	False	False	True	Switch	Delete	Map	View

	Variable	Value
1	Part Name	DAC
2	isDone	False
3	Component	Section
4	x	694
5	y	-401
6	w	228
7	h	220
8	Angle From Origin	-30.0197428784325
9	Orientation From Origin	TOP_RIGHT
10	C Low TH binary	0
11	C High TH binary	255
12	C Low TH Edges	0.01
13	C High TH Edges	0.3
14	C Filtering	fastNlMeans
15	C Morphology	erode dilate
16	Sign	Section
17	hasSign	True
18	xs	-71
19	ys	71
20	ws	25
21	hs	28
22	Angle From Component	135.0
23	Orientation From Component	BOTTOM_LEFT
24	S Low TH binary	0
25	S High TH binary	255
26	S Low TH Edges	0.1
27	S High TH Edges	0.2
28	S Filtering	fastNlMeans
29	S Morphology	
30	Writing	Section
31	hasWriting	True
32	W Low TH binary	0
33	W High TH binary	255
34	W Low TH Edges	0.01

bilateral

Diameter of each pixel neighborhood: 1

Sigma Color: 0,00

Sigma Space: 0,00

☐ Use Binary Images for Edge detection

DAC

Figure 4.28: Example of finished board processing.

Users can then be prompted to save the board and component configurations into a data file, a PKL file, by clicking the 'Save PKL file' button. This action initiates storing the board configuration file under a designated folder called 'board' in the application directory.

Figure 4.29: Button interface to save board data

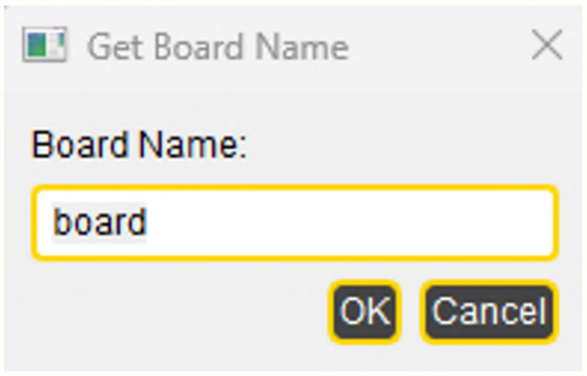


Figure 4.30: Prompt to save board name

When this process is initiated, the program goes through a thorough validation phase to ensure that all critical parameters and configurations have been correctly set and documented.

This includes confirming that all components have been adequately processed `isDone == True`, labelled the three reference points, and recorded essential metadata such as the colour ranges of the PCB board palette.

Once these prerequisites are confirmed, the program proceeds to perform detailed spatial relationship calculations between each component and the three reference points.

Detailed Spatial Relationship Calculations

The procedure involves calculating several key spatial metrics for each component relative to the three reference points:

- **Absolute Distance Calculation:** The function computes the Euclidean distance to each of the three reference points for every component. This metric is essential for establishing a component's exact position on the board.
- **Proportional Distance Calculation:** The function also determines the ratio of each absolute distance to the total sum of all distances from the component to the reference points. This ratio helps normalise the component's position across board images of varying sizes.
- **Directional Distances:** Additionally, the absolute distances in both the x (horizontal) and y (vertical) directions from each component to all three reference points are computed. This information aids in fine-tuning the component's alignment during automated placement or inspection processes.
- **Component Dimensions and Central Coordinates:** The width, height, and central coordinates (x and y) of each component are saved. These dimensions are critical not only for identifying and verifying components but also for precise placement on the PCB.

- **Directional Signs:** The function records the directional sign (positive or negative) of each distance measurement, indicating whether the component is located to the left/right or above/-below each reference point. This detail enhances the accuracy of component placement relative to the reference points.

Top-Level Structure of PKL file

- `iOriginalBoard`: Stores the original golden template image of the board, which is used as a reference for alignment and inspection.
- `colorRangesBoard`: Contains the colour range data for the board's background in BGR, HSV, and LAB colour spaces to assist in component detection and isolation.
- `boardComponents`: A dictionary where each key is a component's name, and the value is another dictionary containing detailed information about that component.
- `componentMap`: Holds the relative coordinates of each component from a defined origin, helping locate components on different board instances.
- `mapComponentsFromReference`: Provides detailed geometric relations between components and predefined reference points on the board to facilitate precise placement.
- `iOriginalNullBoard, xNull, yNull, wNull, hNull`: These fields store information about the board's reference point 1, used as a reference to position other components accurately when drawing on the image.
- `templateDict`: holds all images stored of each component type holding similar components in different positions. The dictionary keys is the component name minus "#" and anything that comes after "#"
  - eg: R#21 and C#1 become $\Rightarrow$ R and C respectively

  Each key holds the data frame of the component type, disregarding its location or size. They will be in the same data frame as long as they serve the same function.

Detailed Component Structure

Each component dictionary in `boardComponents` includes:

- `component`:
  - Status flags like `isDone` to indicate if the component's data collection is complete.
  - Spatial coordinates (`x`, `y`, `w`, `h`) and orientation details relative to the board's reference point 1.
  - Lists of operations performed, such as filtering (`fOperationsComponent`) and morphological transformations (`mOperationsComponent`).

- `componentImages`, `signImages` and `writingImages`: Contains various forms of the components', orientation markers', textual information' images respectively, processed to different extents (e.g., grayscale, binary, edges).
- `sign` and `writing`: Include flags indicating the presence of specific features (like orientation markers or textual information), dimensions and positions of these features relative to the component, and processing details similar to those stored for the component itself.

Inspection Summary Structure

The `inspectionSummary`, `inspectionSummaryYOLO` and `inspectionSummaryTraditionalAndYOLO` dictionaries store aggregate data about the inspection processes conducted using traditional methods, YOLO AI and Traditional plus YOLO AI, respectively. These summaries include:

- Overall statistics about the boards inspected (e.g., total number, defect rates).
- Detailed breakdowns by component regarding defect types and counts.
- Operational metrics such as processing times and individual component analyses.

Component Cross-Referential Data

`mapComponentsFromReference` provides a complex set of measurements that connect each component with three designated reference points on the board. This includes direct distances, proportional distances relative to the sum of all reference point distances, and directional indicators that provide a comprehensive spatial context within the board's layout.

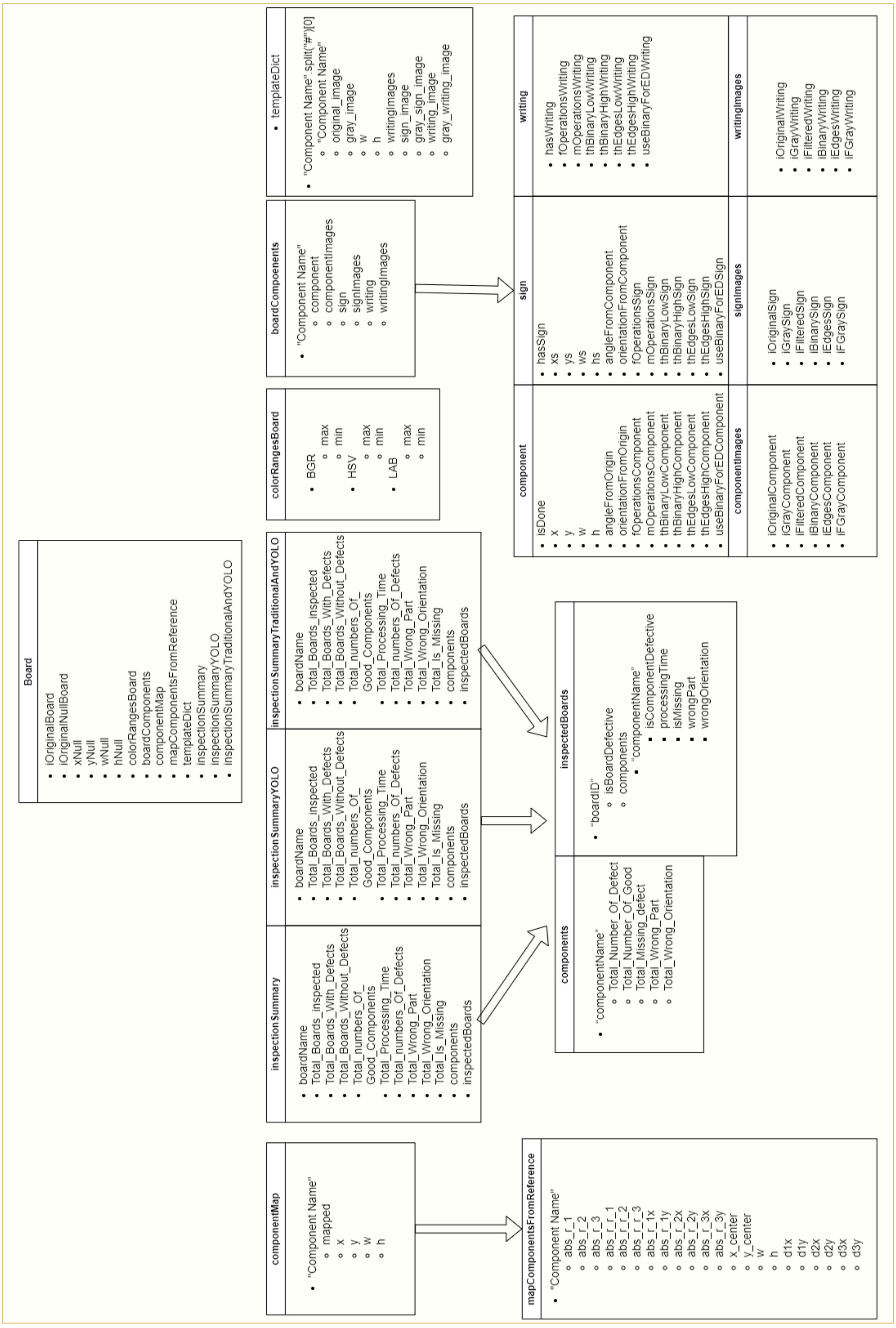


Figure 4.31: Data Structure.

4.4 Dataset preparation and YOLO model training

The following section, provided by Helen Dgheim, discusses how the YOLO AI model was trained for PCB inspection, outlining the dataset preparation, training methods, and results.

4.4.1 Dataset Preparation

To prepare the dataset, we utilised 2 PCB images - one PCB image from BMS-Elektronik and another PCB image of Raspberry Pi from [75] - for the first training method, which was expanded to around 5348 images with simulated defects. As for the second training method, we used 14 high-quality PCB images provided by BMS-Elektronik. These images were divided into smaller sub-images and totalled 546 images.

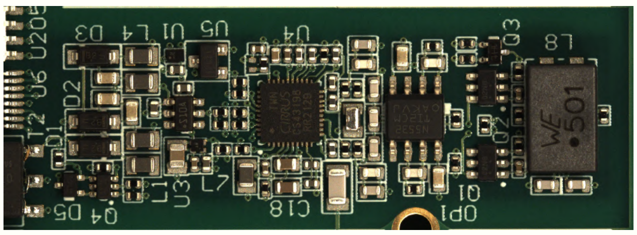


Figure 4.32: Golden Board from BMS-Elektronik

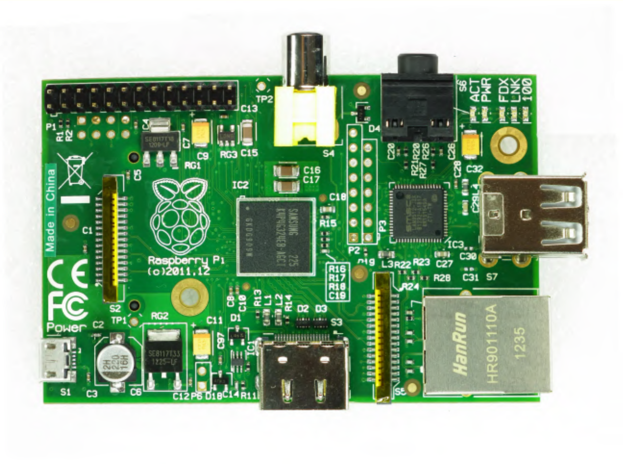


Figure 4.33: High Quality Raspberry Pi
source: [75]

Using the board labelling and analyser tool, data was extracted and labelled automatically in the YOLO-OBB format:

- Format: `class, x, y, x, y, x, y, x, y`
- The dataset was then divided into training and validation sets.

4.4.2 Training Methods

Three distinct training methods were employed to optimise the YOLO model:

- **First Training Method**:
  - A total of 5348 images with simulated defects were used to train a YOLOv8l-OBB model.

- Initially, the hyperparameter ”imgsz” was equal to 640. Therefore, the original images (2448 x 869 pixels) were resized to 640 x 228 pixels, leading to data loss. ”imgsz” is one of the most crucial hyperparameters.
- All class weights were kept equal during training.
- Other hyperparameters such as the number of epochs, learning rate, batch size, and optimiser were adjusted to find the optimal configuration.
- later, we made ”imgsz” equal to 2048 to minimise data loss.

- **Second Training Method**:
  - The 14 high-quality PCB images were divided into multiple sub-images of 640 x 640 pixels, resulting in a dataset of 546 images (454 for training and 92 for validation).
  - The weights for the classes ”reference 1”, ”reference 2”, and ”reference 3” were increased by 3, 2, and 2, respectively, due to the classification issues with ”reference 1” and the importance of the other references.

4.4.3 Results of Training

The Training results provide the following metrics: Box loss, classification loss, Distance Focal Loss (DFL), precision, recall, and mean average precision (mAP) across training epochs for both training and validation datasets. Each graph provides a detailed overview of the model's performance under different conditions.

1. **Box, Classification, and Distribution Focal Loss (Train and Validation)** The Boxloss graphs display the losses for bounding box prediction (train/box_loss and val/box_loss) and classification (train/cls_loss and val/cls_loss). Box loss evaluates the accuracy of locating an object's centre and fitting a bounding box, while classification loss assesses the model's ability to identify the correct object class. A consistent decline in these losses indicates improved object localisation and classification accuracy.

   The distribution focal loss (train/dfl_loss and val/dfl_loss) addresses imbalanced data by focusing more on difficult-to-identify objects, showing how well the model handles objects of varying detection difficulty.

2. **Precision and Recall Metrics**

   The precision (metrics/precision(B)) and recall (metrics/recall(B)) graphs display these metrics over epochs. Precision, the percentage of correct positive predictions, reflects the model's ability to minimise false positives. Recall that the ratio of true positives to actual positives demonstrates the model's effectiveness in identifying relevant instances in the dataset.

3. **Mean Average Precision**

   Mean Average Precision (mAP) measures the model's performance by summarising precision and recall into a single value. Metrics such as mAP50 and mAP50-95 show mAP at different

Intersections over Union (IoU) thresholds, providing a comprehensive evaluation of the model's object detection capabilities across various overlap criteria.

4. **F1-Confidence and Precision-Recall Curves**

   The F1-confidence and precision-recall curves further validate the model's performance. The F1 score combines precision and recall into a single metric, reflecting a balance between identifying positive instances and minimising false positives. A high F1 score indicates the model's excellence in both aspects. The precision-recall curve shows the relationship between precision and recall across different confidence thresholds, illustrating the model's consistency in identifying relevant instances.

- **First Method**:
  - The component recognition was good. No weights were given to the classes.
  - The hyperparameter "imgsz" was changed to 2048 to minimise downscaling. This approach successfully identified all components.
  - this method achieved a mAP of 99.5%.

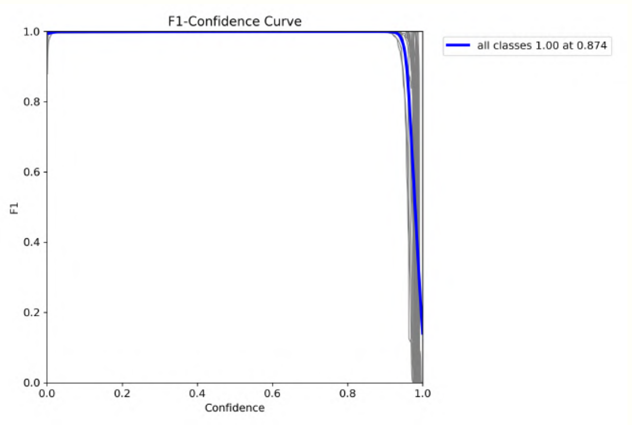


Figure 4.34: F1 Curve Model 1

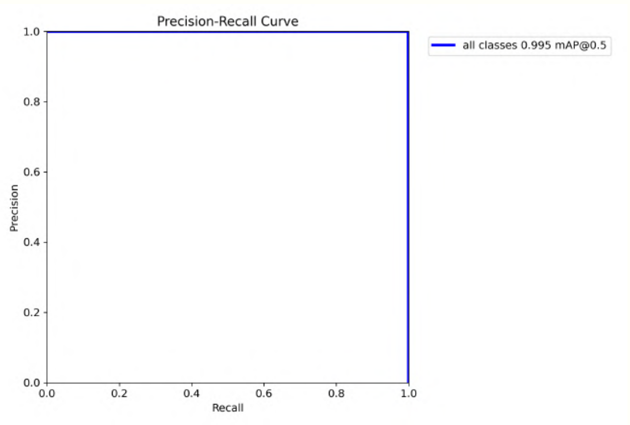


Figure 4.35: Precision-Recall Curve Model 1

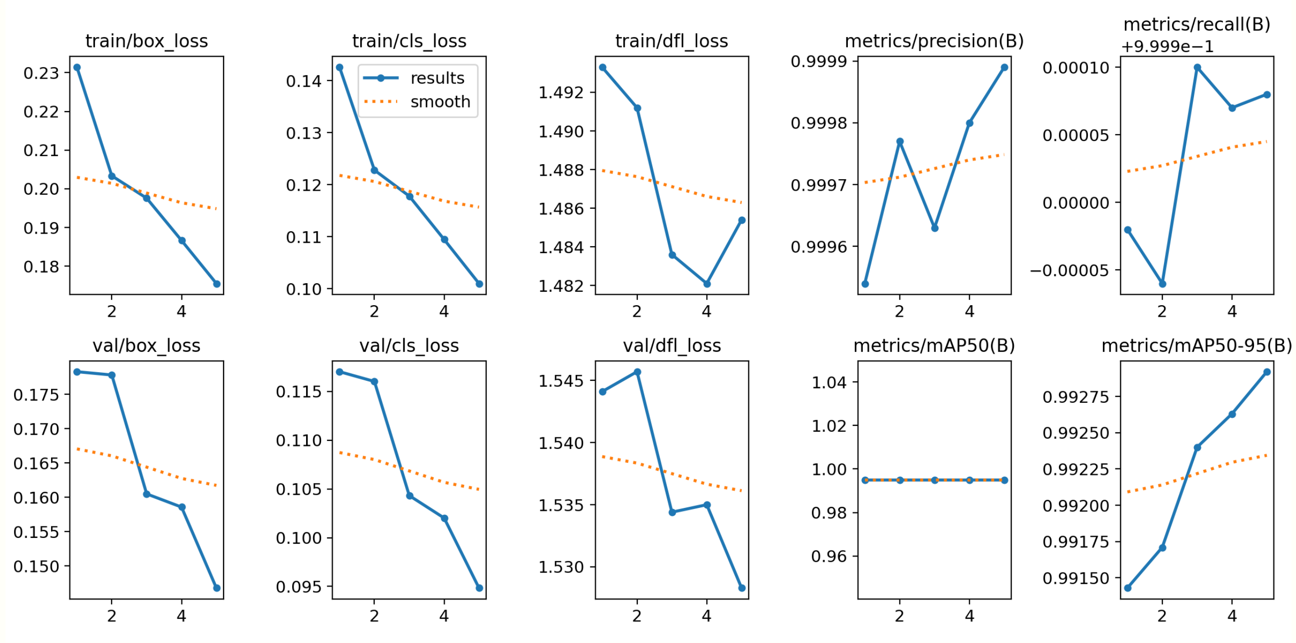


Figure 4.36: Results Model 1

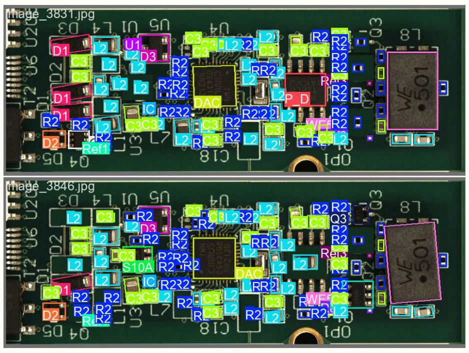


Figure 4.37: Validation Batch Labels Model 1

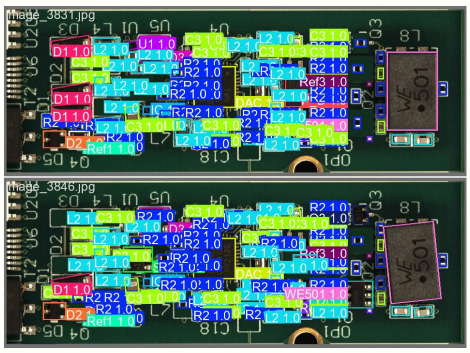


Figure 4.38: Validation Batch Predictions Model 1

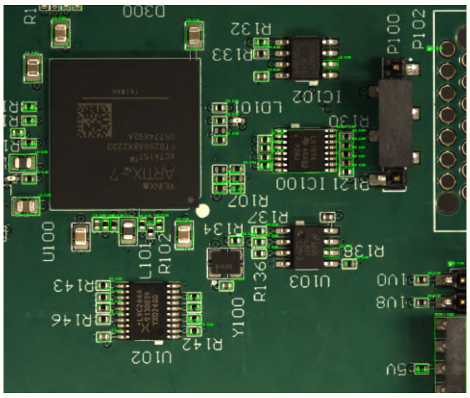


Figure 4.39: Testing AI model 1 with confidence 0.8

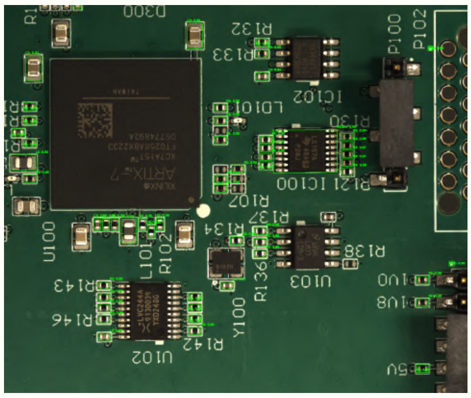


Figure 4.40: Testing AI model 1 with confidence 0.9

- **Second Method**:
  - Taking a full image and then dividing it into 640 x 640 sub-images prevented data loss during training, yielding satisfactory results.
  - Achieved an mAP5 of 99.5%, indicating good performance.
  - During inspection, the board image is divided into multiple 640 x 640 sub-images, predictions are made, and the results are stitched together to reconstruct the larger image before post-processing.

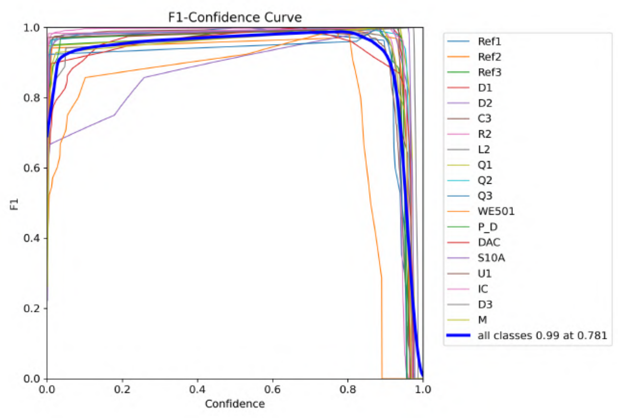


Figure 4.41: F1 Curve Model 2

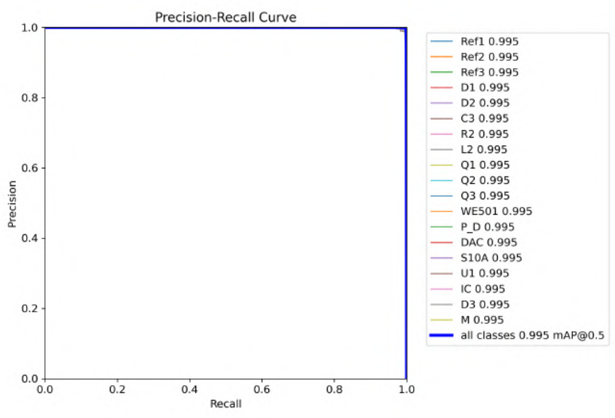


Figure 4.42: Precision-Recall Curve Model 2

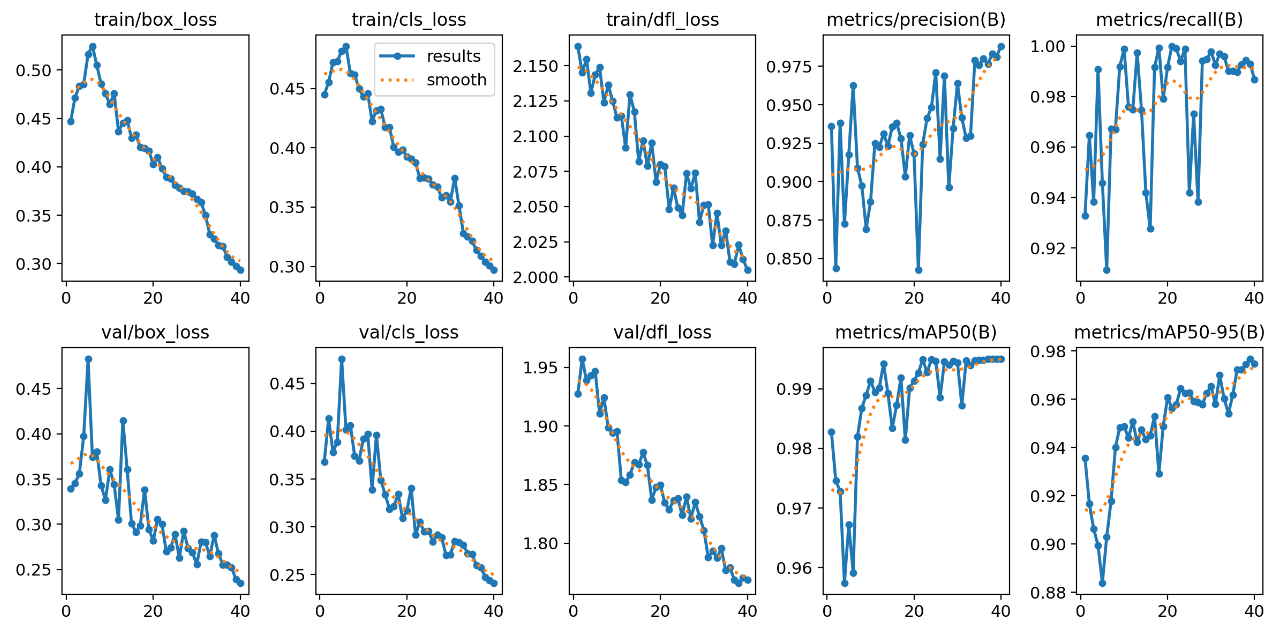


Figure 4.43: Results Model 2

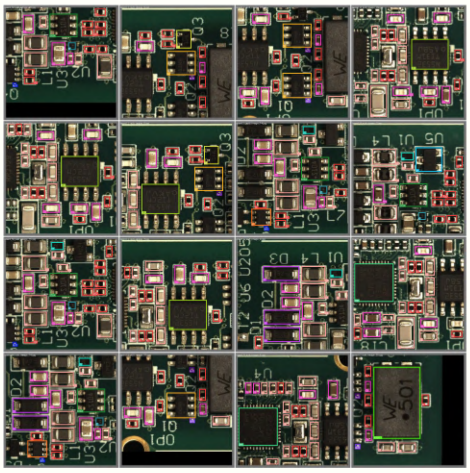


Figure 4.44: Validation Batch Labels Model 2

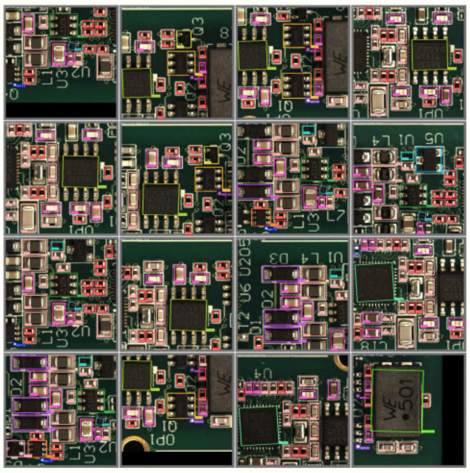


Figure 4.45: Validation Batch Predictions Model 2

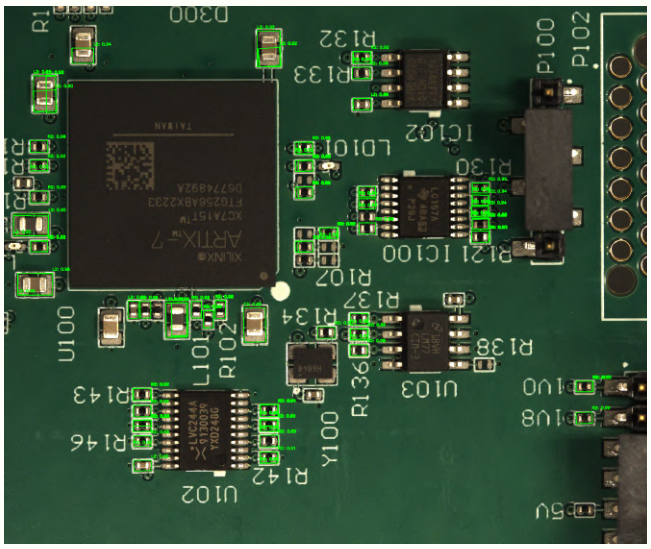


Figure 4.46: Test image demonstrating the annotation results of the second model AI training.

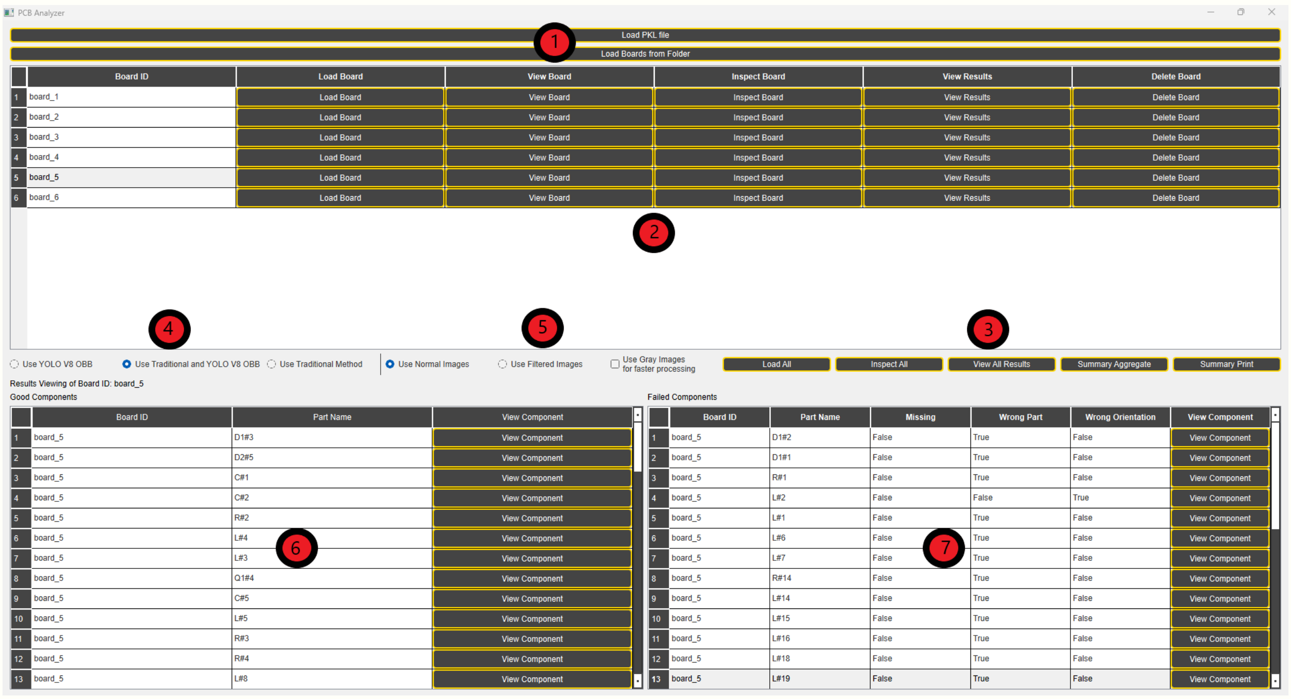
In both AI models, most components were correctly identified when testing the model on brand-new photos, which had a confidence threshold of 0.9. Naturally, some components that were not part of the training set were not classified as expected. At a lower confidence threshold of 0.8, all trained components were successfully detected in the test image, but this also led to the misclassification of some untrained components due to their similar features to the trained ones.

Both AI models provided satisfactory results, which were acceptable for integration into the inspection system. The first model was more favourable.

4.5 Inspection Process Tool

Following the comprehensive labelling of all components and preparation of the golden board template, the `.pkl` file is imported into the second software, ”PCB Inspection Software.”

4.5.1 Overview of the PCB Inspector User Interface



	Board ID	Load Board	View Board	Inspect Board	View Results	Delete Board
1	board_1	Load Board	View Board	Inspect Board	View Results	Delete Board
2	board_2	Load Board	View Board	Inspect Board	View Results	Delete Board
3	board_3	Load Board	View Board	Inspect Board	View Results	Delete Board
4	board_4	Load Board	View Board	Inspect Board	View Results	Delete Board
5	board_5	Load Board	View Board	Inspect Board	View Results	Delete Board
6	board_6	Load Board	View Board	Inspect Board	View Results	Delete Board

Use YOLO V8 OBB [✓ Use Traditional and YOLO V8 OBB] Use Traditional Method | [✓ Use Normal Images] Use Filtered Images | Use Gray Images for faster processing

Load All | Inspect All | View All Results | Summary Aggregate | Summary Print

Results Viewing of Board ID: board_5

Good Components

	Board ID	Part Name	View Component
1	board_5	D1#3	View Component
2	board_5	D2#5	View Component
3	board_5	C#1	View Component
4	board_5	C#2	View Component
5	board_5	R#2	View Component
6	board_5	L#4	View Component
7	board_5	L#3	View Component
8	board_5	Q1#4	View Component
9	board_5	C#5	View Component
10	board_5	L#5	View Component
11	board_5	R#3	View Component
12	board_5	R#4	View Component
13	board_5	L#8	View Component

Failed Components

	Board ID	Part Name	Missing	Wrong Part	Wrong Orientation	View Component
1	board_5	D1#2	False	True	False	View Component
2	board_5	D1#1	False	True	False	View Component
3	board_5	R#1	False	True	False	View Component
4	board_5	L#2	False	False	True	View Component
5	board_5	L#1	False	True	False	View Component
6	board_5	L#6	False	True	False	View Component
7	board_5	L#7	False	True	False	View Component
8	board_5	R#14	False	True	False	View Component
9	board_5	L#14	False	True	False	View Component
10	board_5	L#15	False	True	False	View Component
11	board_5	L#16	False	True	False	View Component
12	board_5	L#18	False	True	False	View Component
13	board_5	L#19	False	True	False	View Component

Figure 4.47: PCB Inspection User Interface

The user interface for the PCB inspection software is designed for simplicity and functionality, enabling efficient inspection of boards. The interface’s key features are outlined below:

1. **Loading Files** This section provides buttons for loading both the golden board **.pkl** file and the individual boards that are to be inspected.

2. **Inspection Table** In this table, each board to be inspected is listed with specific options for:

    - Loading the board into the software
    - Viewing the board’s configuration
    - Initiating the board inspection
    - Viewing the inspection results
    - Removing the board from the table

3. **Batch Processing** This set of controls facilitates batch processing by providing options to:

- Load all boards simultaneously
- Inspect all boards in one go
- Display all results collectively
- Aggregate a summary of the inspection results into the **‘.pkl‘** file and save it to a designated directory
- Print the summary to the console

4. **Inspection Methods** There are three inspection methods available:
    - **YOLOv8-OBB**
    - **Combined Approach** - Both traditional methods and YOLOv8-OBB
    - **Traditional Technique**
5. **Image Types for Inspection** Users can specify the type of images to inspect:
    - Normal images
    - Filtered images
    - Grayscale image either Normal or Filtered
6. **Good Components Table:** This table lists components deemed defect-free by the inspection software. Each can still be viewed individually using the ”View Component” button.
7. **Defective Components Table:** This table displays defective components, with detailed information about the specific issues detected. Similar to the good components, each defective component can be viewed individually.

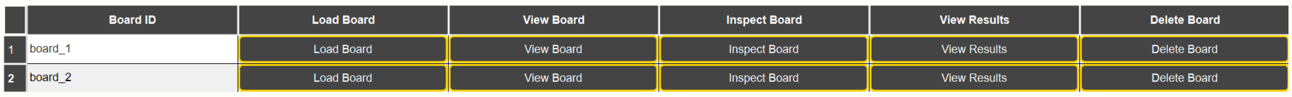
4.5.2 Process Overview: Loading and Preprocessing PCBs for Inspection

Loading PCB Boards for Inspection

Before initiating the inspection process, it is crucial to load the golden board reference **.pkl** file into the software. This file contains all the data prepared through the previous interface, the **PCB analyser and Labeller**, including the defect-free golden board image, board background Colour data, coordinates of the reference points, and mapping of components relative to these reference points. By loading the **.pkl** file, the software initialises a comprehensive summary data structure that will document the inspection results of all processed boards.

Loading the inspection board:

Triggering the "Load Boards from Folder" button imports the PCB boards into the software, generating a data structure for each board based on its ID and populating the inspection table. The table is populated with various actions that can be triggered through dedicated buttons.



	Board ID	Load Board	View Board	Inspect Board	View Results	Delete Board
1	board_1	Load Board	View Board	Inspect Board	View Results	Delete Board
2	board_2	Load Board	View Board	Inspect Board	View Results	Delete Board

Figure 4.48: Boards to be inspected table

Workflow for Board Loading and Preprocessing

Pressing the "Load Board" button initiates an image pre-processing routine that identifies the reference points defined in the golden board file and locates their positions on the board to be inspected. These reference points are used to adjust the board to its correct orientation.

1. **Image Preprocessing: Rotation Alignment**

   • The accurate alignment of the inspected board is achieved by referencing the three predefined points, ensuring that all subsequent comparisons and analyses are based on a properly oriented template. Below is an example of this alignment process:

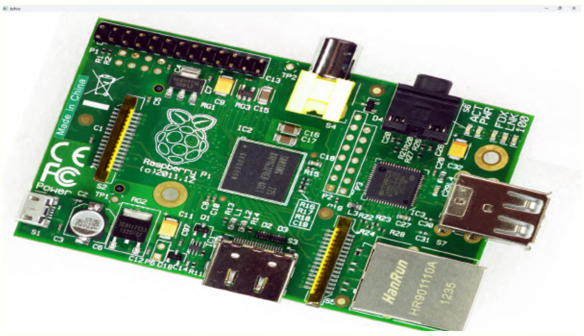


Figure 4.49: Example 1: before rotation alignment

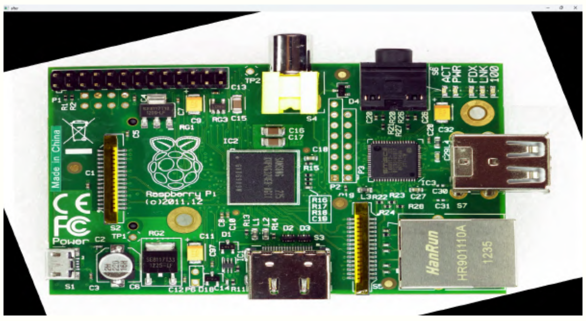


Figure 4.50: Example 1: after rotation alignment

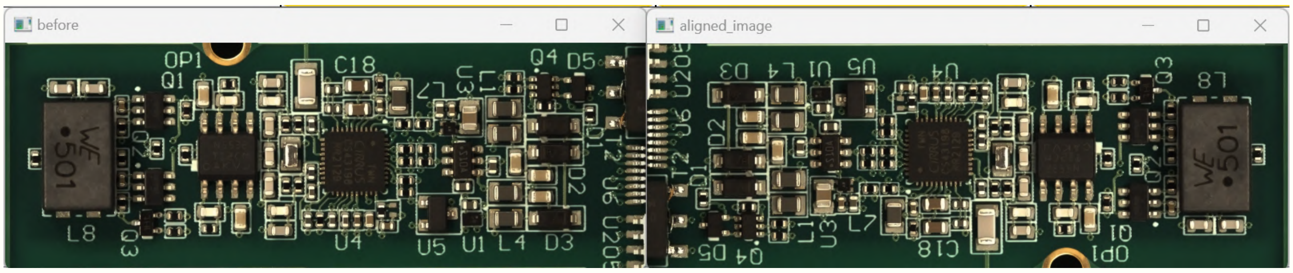


Figure 4.51: Example 2: before (left) and after (right) rotation alignment

- **Importance of Reference Points** Identifying these reference points is important for the correct alignment of the inspected board to the golden template. The method ensures that the PCB orientation remains consistent and correctly oriented throughout the inspection process.

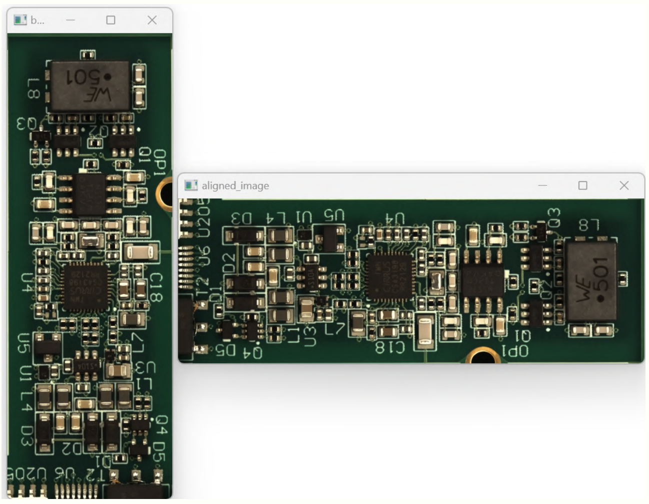


Figure 4.52: Example 3: before (left) and after (right) rotation alignment

2. **Reference Data Extraction and Coordinate Calculation Process**

    - **Extraction and Loading of Reference Data:** Initially, reference points pre-defined within the board components are extracted and respectively arranged in a reference list. These serve as key anchors for accurate positioning and inspection. The data includes the distance between each other and the coordinates of their positions x and y relative to the model image.

    - **Application of Template Matching:** A template matching algorithm is applied to locate these reference points on the PCB. The algorithm scans the entire board to identify potential matches for each reference. Given that similar patterns may occur, exhaustive scanning is essential to prevent misidentification. For instance, the algorithm may detect numerous potential matches (e.g., 10, 100, and 9000 matches for the first, second, and third references, respectively).

    - **Filtering and Verification of Reference Matches:** Postdetection, the challenge is to distinguish the correct set of reference points among the multitude of potential matches. This filtering is managed through a function, `find_correct_matches`, which utilises pre-existing geometric relationships established during the setup phase:

        – The function takes in a list of all matched coordinates and compares them against pre-calculated distances using the Euclidean formula.

        – Cartesian differences (delta x and delta y) are computed between every possible pair across the three references.

  - The distances are compared against the known distances, isolating the correct combination of three points (one from each reference category) whose relative distances align with pre-established values.

```python

def find_correct_matches(self,matches,saved_distances):
    correct_combinations = []
    x = [np.array(matches[i])[:, 0] for i in range(3)]
    y = [np.array(matches[i])[:, 1] for i in range(3)]
    distances = []
    for i, (a, b) in enumerate([(0, 1), (0, 2), (1, 2)]):
        dist_x = x[b].reshape(1, -1) - x[a].reshape(-1, 1)
        dist_y = y[b].reshape(1, -1) - y[a].reshape(-1, 1)
        dist = np.stack((dist_x, dist_y), axis=-1)
        distances.append(np.abs(np.linalg.norm(dist, axis=-1) -
   saved_distances[i]))
    a, b = distances[0].shape
    _, c = distances[1].shape

    dist = np.repeat(np.expand_dims(distances[0], 2), c, axis=-1)
    dist += np.repeat(np.expand_dims(distances[1], 1), b, axis=1)
    dist += np.repeat(np.expand_dims(distances[2], 0), a, axis=0)
    indices = np.unravel_index(np.argmin(dist), dist.shape)

    correct_combinations.append((matches[0][indices[0]], matches[1][
   indices[1]], matches[2][indices[2]]))

    return correct_combinations,indices

```

The function reduces the computational time needed to identify the correct references from 10 minutes to less than a second.

- **Final Adjustments and Alignment:**
  - Upon successfully identifying the true reference points, the PCB image may require further adjustments—specifically, translation and rotation—to align perfectly with the 'golden' template used for reference. These adjustments are calculated to match the exact orientation and position relative to the reference points in the original template, ensuring uniformity regardless of the initial state of the PCB. Then, another template matching is applied to find the new reference point coordinates from the newly aligned image.
- **Coordinate Calculation for Component Positioning**
  For each component, the system calculates the x and y coordinates relative to each of the three reference points using the known distances and directional indicators stored in the data structure. The process includes:
  - **Distance Measurement:** For each reference point, the horizontal (x) and vertical (y) distances from the component are calculated.

- **Directional Adjustment:** These distances are adjusted based on the directional data (`d1x`, `d1y`, etc.), which indicate the relative position of the component concerning each reference point.

```

# Example calculations for x and y coordinates relative to each
   reference point
centerxr1 = self.r1x + (abs_r_1x*d1x)
centeryr1 = self.r1y + (abs_r_1y*d1y)
#...

```

- Here, `r1x` and `r1y` are the coordinates of the reference point, `abs_r_1x` and `abs_r_1y` represent the distance from the reference point, and d1x and d1y indicate the directional vectors from the reference point to the component.

3. **Validation and Extraction of Component Images**

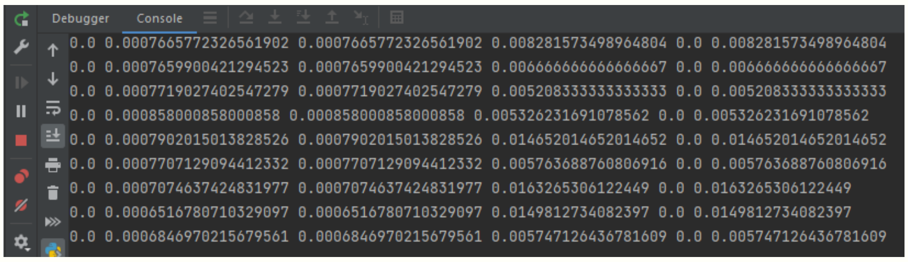
   After computing the coordinates of each component relative to all three reference points, their positional consistency is evaluated. This involves comparing the differences between the computed x and y coordinates from each reference point:

```

delta1 = abs(centerxr1 - centerxr2)/((centerxr1 + centerxr2)/2)
delta2 = abs(centerxr2 - centerxr3)/((centerxr2 + centerxr3)/2)
delta3 = abs(centerxr1 - centerxr3)/((centerxr1 + centerxr3)/2)
delta4 = abs(centeryr1 - centeryr2)/((centeryr1 + centeryr2)/2)
delta5 = abs(centeryr2 - centeryr3)/((centeryr2 + centeryr3)/2)
delta6 = abs(centeryr1 - centeryr3)/((centeryr1 + centeryr3)/2)

```

In practice, positional discrepancies are typically less than 2%, but a higher threshold of 10% is employed for additional safety. This ensures that variations among all three reference points remain within acceptable bounds.



```
0.0 0.0007665772326561902 0.0007665772326561902 0.008281573498964804 0.0 0.008281573498964804
0.0 0.0007659900421294523 0.0007659900421294523 0.006666666666666667 0.0 0.006666666666666667
0.0 0.0007719027402547279 0.0007719027402547279 0.005208333333333333 0.0 0.005208333333333333
0.0 0.000858000858000858 0.000858000858000858 0.005326231691078562 0.0 0.005326231691078562
0.0 0.0007902015013828526 0.0007902015013828526 0.014652014652014652 0.0 0.014652014652014652
0.0 0.0007707129094412332 0.0007707129094412332 0.005763688760806916 0.0 0.005763688760806916
0.0 0.0007074637424831977 0.0007074637424831977 0.0163265306122449 0.0 0.0163265306122449
0.0 0.0006516780710329097 0.0006516780710329097 0.014981273408239 7 0.0 0.0149812734082397
0.0 0.0006846970215679561 0.0006846970215679561 0.005747126436781609 0.0 0.005747126436781609
```

Figure 4.53: Delta value difference between components

The differences between calculated positions (`delta values`) are carefully checked to ensure consistent positioning across all references. If the positional consistency checks are met, the corresponding component image is extracted from the inspected board image.

The extracted image is not limited to the component's immediate size to account for potential errors and ensure comprehensive coverage. Instead, a square region is captured, extending outward to include the component's diagonal plus the maximum detected positional error. This approach guarantees that the extracted image fully captures the component, even if slight deviations exist in reference point calculations.

4. **Storing Component Data**
   Each component's extracted image and metadata are organised into a structured format, typically a dictionary. The metadata includes the component's dimensions, relative orientation, and position relative to the reference points.

   - **Traditional Method Inspection Data:** For traditional inspection, the component image is stored directly in the data structure, ensuring that relevant information like coordinates and dimensions is captured.

```
self.inspectionBoard[boardID]["parts"][key] = {
    "iOInspectionPart": cropped_img,
    "iOGrayInspectionPart": cv2.cvtColor(cropped_img,cv2.
   COLOR_BGR2GRAY),
    "isMissing": False,
    "wrongOrientation": False,
    "wrongPart": False,
    "isFound": False,
    "x":x_s-self.r1x+int((x_e-x_s)/2),
    "y":y_s-self.r1y+int((y_e-y_s)/2),
    "w":x_e-x_s+xdeltamax,
    "h":y_e-y_s+ydeltamax,
    "orientation": None }

```

   - **YOLO-Only Method Inspection Data:** For YOLO-based inspections, the structure is created without storing the actual component image. The YOLO v8 approach detects all components and reference points in the image and then verifies their positions relative to the reference points.

```
self.inspectionBoard[boardID]["YOLO"]["parts"][key] = {
    "iOInspectionPart": None,
    "iOGrayInspectionPart": None,
    "isMissing": False,
    "wrongOrientation": False,
    "wrongPart": False,
    "isFound": False,
    "x": None,
    "y": None,
    "w": None,
    "h": None,
    "orientation": None }

```

- **Traditional and YOLO Combined Method Inspection Data:** For a combined traditional and YOLOv8-OBB inspection, the component image is stored in addition to the coordinates of the components similar to the traditional technique.

```
self.inspectionBoard[boardID]["TraditionalAndYOLO"]["parts"][key] =
    {
     "iOInspectionPart": cropped_img.copy(),
     "iOGrayInspectionPart": cv2.cvtColor(cropped_img, cv2.
    COLOR_BGR2GRAY),
     "isMissing": False,
     "wrongOrientation": False,
     "wrongPart": False,
     "isFound": False,
     "xc": centerxr1,
     "yc": centeryr1,
     "x": x_s - self.r1x + int((x_e - x_s) / 2),
     "y": y_s - self.r1y + int((y_e - y_s) / 2),
     "w": x_e - x_s,
     "h": y_e - y_s,
     "orientation": None }

```

Each component, associated with its board ID, is mapped to the reference points, and each inspection method has its own data structure. This enables comparative analysis between the different approaches.

Pre- and Post-inspection Visualisation

After the board is loaded and pre-processed, it can be visually inspected prior to starting the actual inspection by clicking the ”View Board” button associated with the board ID. This preview provides a preliminary view of component positioning as identified during pre-processing.

Visual Representation Before Inspection

The board image is presented with highlighted regions representing the expected component locations. These highlights are determined from pre-calculated data, giving an early overview of the expected placements. However, they are not final validations of component presence or orientation, and further inspection is necessary.

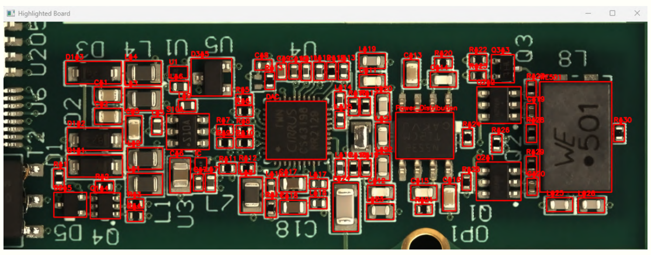


Figure 4.54: Board Pre Inspection

Visual Representation After Inspection

This post-inspection visualisation provides a more comprehensive view of the inspected board. The exact component positions are verified, and any defects are identified and highlighted. This view gives a clear comparison between pre-inspection expectations and the actual results after the detailed inspection.

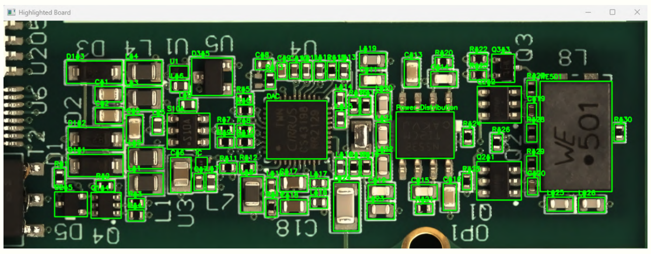


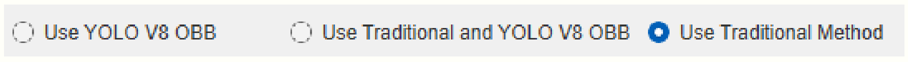
Figure 4.55: Board Post Inspection

Inspection Methodologies

Here, we will discuss the definition of each inspection method before going into the inspection process in detail. To trigger an inspection, you need to press the “inspect board” button associated with the board on the board’s table.

The inspection of PCBs can be conducted using various methods:

- **Traditional Inspection:** Utilises conventional image processing techniques to identify defects.
- **YOLO AI Inspection:** Employs the YOLO V8 Object Bounding Box model to automate component detection. However, some post-processing is needed to identify the defects and verify the component location.
- **Traditional and YOLO AI Inspection:** utilises both advantages of the traditional and AI techniques. Where it determines the location of the components using traditional techniques and verifies their class or type using AI by passing the full image of the board to the AI model. Then, for reassurance, match it with the golden component reference.



Use YOLO V8 OBB   Use Traditional and YOLO V8 OBB   [✓ Use Traditional Method]

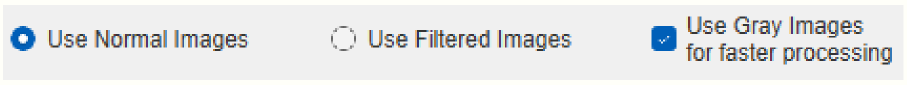
Figure 4.56: User interface inspection methods

Choice of Imaging Techniques

Depending on the inspection environment and equipment quality, different imaging techniques may be employed:

- **High-Quality Cameras:** If the capture device produces high-resolution images, filtering might not be necessary.

- **Variable Lighting Conditions:** Grey-scale and filtered images are preferred under fluctuating lighting conditions to maintain consistency in image quality.
- **Speed Requirements:** For rapid processing, using grey-scale images can significantly reduce computation time.



[✓ Use Normal Images] Use Filtered Images [✓ Use Gray Images for faster processing]

Figure 4.57: User interface image type

Inspection Results Examination

After completing the inspection process, results can be accessed through the program interface:

- **Results Overview:** Users can view the inspection results by selecting the respective board. Defect-free components are listed in one table, while components with detected issues are listed in another.

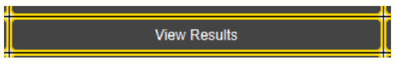
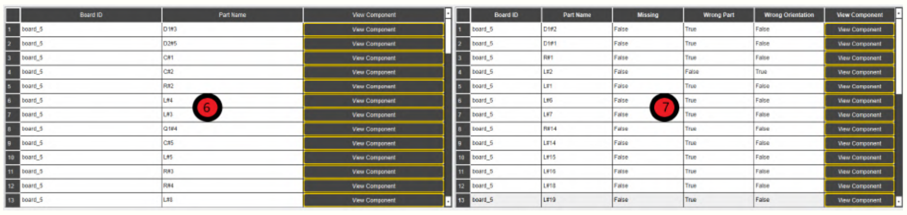


Figure 4.58: View result button

- **Detailed Component View:** By clicking on the "view component" button, users can compare a specific component against its golden reference to assess discrepancies.



	Board ID	Part Name	View Component
1	board_5	D1#3	View Component
2	board_5	D2#5	View Component
3	board_5	C#1	View Component
4	board_5	C#2	View Component
5	board_5	R#2	View Component
6	board_5	L#4	View Component
7	board_5	L#3	View Component
8	board_5	Q1#4	View Component
9	board_5	C#5	View Component
10	board_5	L#5	View Component
11	board_5	R#3	View Component
12	board_5	R#4	View Component
13	board_5	L#8	View Component

	Board ID	Part Name	Missing	Wrong Part	Wrong Orientation	View Component
1	board_5	D1#2	False	True	False	View Component
2	board_5	D1#1	False	True	False	View Component
3	board_5	R#1	False	True	False	View Component
4	board_5	L#2	False	False	True	View Component
5	board_5	L#1	False	True	False	View Component
6	board_5	L#6	False	True	False	View Component
7	board_5	L#7	False	True	False	View Component
8	board_5	R#14	False	True	False	View Component
9	board_5	L#14	False	True	False	View Component
10	board_5	L#15	False	True	False	View Component
11	board_5	L#16	False	True	False	View Component
12	board_5	L#18	False	True	False	View Component
13	board_5	L#19	False	True	False	View Component

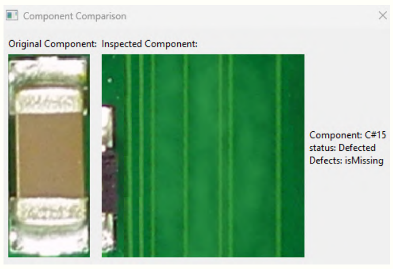
Figure 4.59: Good (left) and bad (right) components tables

4.5.3 Error Classification in PCB Inspection

During the inspection of PCBs, various errors can be identified, which are crucial for ensuring the quality and functionality of the boards. These errors are categorised as follows:

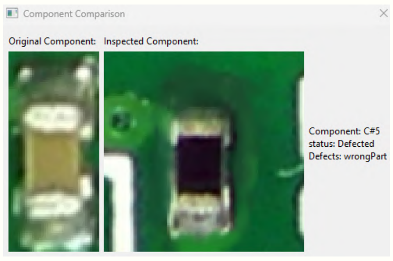
- **Missing Component:** Identified by the "isMissing" flag within the boardID dictionary. This error indicates that a component that should be present on the PCB is absent.

- **Wrong Part:** Denoted by the “wrongPart” flag, this error signifies that a component does not match the model component.
- **Incorrect Orientation:** Marked by the “wrongOrientation” flag, indicating that a component is placed in the wrong orientation.



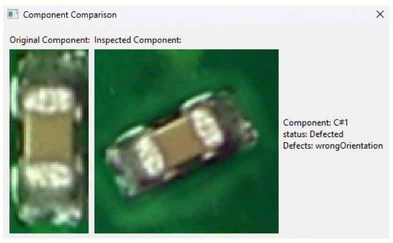
```
Component: C#15
status: Defected
Defects: isMissing
```

Figure 4.60: Missing component



```
Component: C#5
status: Defected
Defects: wrongPart
```

Figure 4.61: Wrong component



```
Component: C#1
status: Defected
Defects: wrongOrientation
```

Figure 4.62: Wrong orientation

These primary error types help in diagnosing other related defects. For instance, a damaged component might be categorised under the ”Wrong Part” error.

4.5.4 Process Overview: Traditional PCB Inspection

This section outlines the detailed procedure followed for the traditional inspection of PCBs using image processing techniques. This method relies on meticulous image analysis and precise condition checks to ensure each component’s integrity.

Initial Setup and Component Loop

Upon loading the board image and selecting the desired inspection method option, the inspection starts by pressing the ’Inspect Board’ button within the table for the targeted board. The inspection process involves several key steps:

1. **Component Loop:**
   Each component expected to be on the PCB is iterated over. At the start of each loop, any failure flags for the current component are reset.
   ```
   self.inspectionBoard[boardID]["parts"][name]["wrongOrientation"] = False
   self.inspectionBoard[boardID]["parts"][name]["wrongPart"] = False
   self.inspectionBoard[boardID]["parts"][name]["isMissing"] = False
   ```

2. **Missing Component Detection:**
   Colour Range Checking: The middle region of the component’s image is analysed to determine if it matches the background colour of the board, indicating a missing component. This involves extracting the mean colour values in the BGR, HSV, and LAB spectrums and comparing these with the predefined background colour ranges using the `is_color_within_range` function.

- If the colour matches the background across all colour spaces, the 'isMissing' flag is set, and the loop skips to the next component as no further processing is needed.

3. **Template Matching and Orientation Verification:**

    - **Initial Template Matching:** The first attempt to locate the component uses the cv2.TM_CCOEFF_NORMED method with an accuracy threshold of 70%. If the component does not meet this threshold, further analysis is conducted by looping through the component template catalogue of the same specific component type stored in templateDict until a match is found. In case a match is found, the program replaces the template image with the one that matched and continues the analysis. In case it is not found, the program uses the original template and the SIFT algorithm to determine if the component is present and has an orientation.
    - **SIFT Analysis:** The `calculate_rotation_angle` function is applied if the initial template matching fails to confirm the component's presence.
        - This function assesses whether the component is found and determines its rotation angle. Components rotated beyond ±2 degrees are flagged as having the wrong orientation.
        - If the component is not found using sift also, then the component is flagged as being the wrong component, "wrong part".

4. **Textual Information and Orientation Markers**
Following the initial verification of component presence and orientation, further checks are conducted to ensure the accuracy of textual information and orientation markers, which are critical for component identification and correct placement
Verification of Textual Information For components expected to contain textual identifiers, has it **"hasWriting"** flag is True:

    - **Template Matching for Text:** Initially, template matching is employed to locate the expected text on the component using a 70% threshold for accuracy. If the text matches the template satisfactorily, the process progresses. In case no match is found, the program uses a sift algorithm.
    - **Fall back to sift:** the SIFT algorithm is invoked to determine if the text can be identified less rigidly. This secondary check helps to confirm the presence of text even under slight variations in appearance or alignment.
        - **Rotation Check for Textual Information:** Upon locating the text, its orientation is assessed. If the text's rotation deviates by more than 2 degrees from the expected alignment, this is flagged as incorrect orientation. This step ensures that the text not only matches visually but is also correctly aligned relative to the component's intended design.

- **Defect Handling for Textual Information:** If neither template matching nor SIFT can confirm the text's presence, it indicates a similar component with a different model name is in place of the correct component, leading to a 'wrong part' classification due to incorrect or missing textual information.

Orientation Marker Verification
For components that include specific orientation markers (e.g., a circle or groove indicating pin 1 positions) **"hasSign"** flag should be True:

- **Marker Identification via Template Matching:** The search for orientation markers begins with template matching. Successful identification confirms the marker's presence and allows further positional checks.
  - **Positional Accuracy Check:** If a marker is identified, its position relative to the component's centre is compared against the expected 'template' positions (e.g., north, northeast, west). This comparison determines if the marker's orientation aligns with the standard configuration for that component type.
  - **Orientation Compliance:**
    - If the marker's orientation mismatches the predefined standard, a 'wrong orientation' flag is raised, indicating that while the marker exists, it does not conform to the required positioning.
    - Absence of a detected marker when one is expected leads to a 'wrong part' flag, as the missing marker implies a different component.

Quick Summary Component

- component image matches PCB background → missing
- template match → found → ok → check textual information
- template match → not found → loop through other component template images of the same type
- template match (from other template images) → found → replace template image → check textual information
- template match (from other template images) → not found → sift algorithm
  - sift → found → check rotation angle
    - rotation angle → bigger than 2 → wrong orientation
    - rotation angle → smaller than 2 → ok → check textual information
  - sift → not found → wrong part

Quick Summary Textual information

- template match → found → ok → check orientation marker

- template match → not found → sift algorithm
  - sift → found → check rotation angle
    - rotation angle → bigger than 2 → wrong orientation
    - rotation angle → smaller than 2 → ok → check orientation marker
  - sift → not found → wrong part

Quick Summary Orientation Marker

- template match → found → ok → check orientation
  - orientation → match → ok
  - orientation → does not match → wrong orientation
- template match → not found → wrong part

Process Outcomes and Flagging

The inspection system records each finding, adjusting flags within the component's data structure to reflect the identified issues:

- `isMissing:` Set if no component is detected in the designated area, indicating a missing part.
- `wrongPart:` Raised if textual information or orientation markers are incorrect or absent.
- `wrongOrientation:` Applied if the component's or text's orientation is outside acceptable limits.

Final Assessments and Iteration

Defect-Free Confirmation: If no defects are detected through these checks, the component is marked as found.
The process repeats for each component on the PCB until all have been inspected.

4.5.5 Process Overview: AI-based Inspection Using YOLO

Integration and Operation of the YOLO Model

The AI-based inspection process within the system leverages the capabilities of the YOLO model, which is adept at processing images to identify and categorise objects within a single viewing. This section delves into how the trained YOLO model is embedded in the program and utilised to analyse the PCB images for defects and component recognition.

Output Format and Data Extraction

When a board image is input into the YOLO model, it generates results in several formats. For inspection purposes, the specific output format extracted is the OBB parameters and class identifiers:

- **OBB Parameters** (`results[0].obb.xywhr`): This includes the centre coordinates (x, y), dimensions (width w, and height h), and the rotation (r in radians) of each detected shape.
- **Class Identifiers** (`results[0].obb.cls[i]`): Each detected shape is associated with a class ID, which helps in identifying the type of component or reference point detected.

The class name, corresponding to the type of object detected, such as different types of PCB components or reference points, is retrieved using the class ID.

Categorisation and Storage of Detection Results

Upon obtaining the results from the AI model, the next step involves categorising and storing these results for further analysis:

- **Reference Points:** Detected items classified as reference points (references 1, 2, and 3) are stored in specifically designated lists (`matches1`, `matches2`, `matches3` for coordinates and `matcheswhr1`, `matcheswhr2`, `matcheswhr3` for dimensions and rotation). This structured approach ensures that the spatial and dimensional data of reference points are maintained in an ordered manner, which is critical for subsequent positional calculations.
- **Other Components:** All other detected components, not classified as reference points, are placed in a list named `self.foundComponents`. Each entry in this list is a dictionary containing detailed attributes of the object found:

```
self.foundComponents.append(
    {"class_name": class_name,
     "x": x, "y": y, "w": w, "h": h, "r": r,
     "abs_r_1":None,"abs_r_2":None,"abs_r_3":None,
     "abs_r_r_1":None,"abs_r_r_2":None,"abs_r_r_3":None,
     "c": confidence})

```

Determining Correct Reference Positions

After the AI model has processed the PCB image and identified potential reference points and components, the next crucial step involves accurately determining the correct positions of these references. This is achieved through a systematic post-processing approach designed to filter and validate the results, ensuring precision in the final analysis.

Post-processing and Validation

The raw outputs from the YOLO model can sometimes include duplicate detections or erroneous data. To refine these results, we employ the previously discussed function, find_correct_matches, which plays a role in discerning the correct reference points among multiple candidates:

1. **Input Preparation:**
   - each centre coordinates (x, y) of potential reference points are grouped according to which reference they refer to and each group is placed into a list as an element, then the list is passed as the first argument to the function find_correct_matches.
   - The second argument is self.saved_distances, which contains the pre-calculated distances between the reference points (e.g., between reference 1 and 2, 1 and 3, and 2 and 3).
2. **Execution and Outcome:**
   - The function evaluates the spatial relationships between all possible combinations of reference points against the known distances to identify the most likely accurate trio of references.
   - This yields the most probable coordinates and indices of the correct reference points.
3. **Updating Reference Information:**
   - With the correct references identified, their coordinates and dimensions are updated in the self.references dictionary to reflect their verified positions and attributes.

```
src_points1 = correct_combinations[0][0]  # Example to retrieve coordinates
whrc1 = matcheswhr1[indices[0]]  # Example to retrieve dimensions and confidence
self.references["Reference1"].update({"x": src_points1[0], "y": src_points1[1], "w": whrc1[0], "h": whrc1[1], "r": whrc1[2]})
self.references["Reference1"]["confidence"]=whrc1[3]

```

Trilateration and Component Positioning

Once the reference points are set, the system uses trilateration methods to position each component relative to these fixed points accurately. This involves updating several metrics for each detected component:

- **Absolute Distances** (abs_r_x): Calculated as the Euclidean distance from the component to each reference point.
- **Relative Ratios** (abs_r_r_x): Represent the proportion of each absolute distance relative to the total distance from all references.

This approach ensures each component's location is precisely defined in relation to the established reference points, enabling further inspections and quality assessments.

Once all preliminary detections are classified and stored appropriately using the AI-based YOLO model, the subsequent step involves validating these detections to ensure their correctness and relevance to the pre-defined component information. This validation process is crucial in minimising errors due to duplicated detections or misidentifications by the AI system.

Setup and Resetting Flags Initial actions for each component that needs to be inspected include resetting any previously set flags that might indicate defects or identification statuses. This ensures that each inspection cycle starts with a clean slate, preventing data carryover issues.

In addition, when starting to loop over each component, the `self.foundComponents` is divided into two lists.

A list that has a class name matching the component name targeted for this iteration, and the other list has all other components that do not match their class name. This will reduce processing time.

Position Matching Process Each component's position relative to the three reference points is critical for ensuring accurate placement on the PCB. This data is already stored in `self.mapComponentsFromReference`, which contains detailed distance and positional information necessary for verifying the localisation done is the same as the localisation stored.

Validating Component Position The core of the validation process involves comparing the detected positions (from `self.foundComponents` which contains shapes detected by YOLO) against the expected positions (from `self.mapComponentsFromReference`). The comparison metrics include:

- **Absolute Distance Differences:** The delta difference between the stored and the calculated in the absolute distance from reference to the component.
  - `deltaR1 = abs(part["abs_r_1"]-component["abs_r_1"])`
- **Relative Distance Ratios:** Differences in the ratio of each absolute distance to the sum of distance length from the component to each reference are also evaluated.
  - `deltaRR1 = abs(part["abs_r_r_1"] - component["abs_r_r_1"])`
- **diagonal size:** The difference in diagonal found of the object with the diagonal of the original component is calculated and then divided by the original diagonal to get the percentage difference.

Post-processing for Final Confirmation

Even if a detected component matches the expected position and size within set thresholds, further verification is necessary to confirm its identity, condition and orientation.

1. **initiating the Inspection**
   first, loop through the list that has matching class names with the target component. Then, if no matches are found, loop through the list with different class names:

- The inspection commences by matching the detected shapes' locations with the expected component positions. The matching is subject to a strict threshold criterion: a maximum delta difference between absolute distances `deltaRx` of 8% of the component diagonal size and a maximum 2% in delta ratio of reference absolute distance `deltaRRx`. The calculations prioritise the ratio of absolute distances to the total distance from all references for improved accuracy.

2. **Verification Stages**

- **Initial Location Verification** If no correct match is found for a location, it indicates a missing component, triggering the setting of a `isMissing` flag within the board's record
- **Component Validation Checks**
  - Concurrent validation checks are performed to ensure:
    - The diagonal difference between the expected and detected components is within allowable limits.
    - The detected component's class name matches the predefined component label.
  - If these conditions are not met (diagonal difference $\geq 20\%$ or mismatched class names), the component is flagged as incorrect `wrongPart`
- **Orientation and Alignment Verification**
  - For components that pass the initial validation, They then go through template matching. If matched using template matching, then the component is found, and the software should move on to the next component.
  - If not matched, they then go through the SIFT algorithm to determine any rotational discrepancies:
    - If a rotation deviation greater than 2 degrees is detected, it is flagged as a potential orientation error `wrongOrientation`.
    - If no match is found using Sift, then the component is then the wrong part `wrongPart`
- **Final Determinations**
  - Conversely, if all checks are satisfied—accurate location, correct class name match, and acceptable orientation—the component is confirmed as correctly placed:
    - Successfully identified components are then removed from the processing list to optimise subsequent detection cycles.

Rationale Behind Using SIFT for Orientation Verification

Although YOLO provides rotational data (`r`, in radians), its precision for orientation can sometimes fall short of the requirements for high-accuracy component placement. The SIFT algorithm is employed as an additional verification step to counteract potential errors and ensure each component's precise orientation.

Summary of the AI-Based Inspection Process

The inspection process using the AI model ensures precise component placement and verification on PCBs:

- **Results Retrieval**: Initially, the AI model outputs detection results for further processing.
- **Reference Localisation**: Utilising triangulation, the exact locations of three key reference points are determined.
- **Distance Calculation**: Absolute distances and other relevant localisation details of each component relative to these references are computed.
- **Position Matching**: The detected shapes are matched against the expected component positions.
  - If no accurate match is found, it is concluded that the component is missing `isMissing`.
- **Dimension and Class Verification**: For shapes that match location-wise:
  - The diagonal dimensions and class names of the shapes are compared to those of the expected components.
  - If there is a discrepancy in size or class, the item is identified as the incorrect component `wrongPart`.
- **Template matching**: if the object matched the template, then there is no need for further processing, and that means the component was found
- **Orientation Check**: in case the template match fails, the program uses the SIFT algorithm to find component presences and check if the orientation of the matched components is correct.
  - A rotation difference greater than 2-degree flags the component as having incorrect orientation `wrongOrientation`.
  - If the orientation matches within acceptable limits, the component is confirmed as correctly found.
  - If the SIFT algorithm fails to confirm the component's presence, it is marked as the wrong component `wrongPart`.

4.5.6 Process Overview: Traditional and AI-Based Combined Inspection Method

Traditional and AI-based inspections each offer distinct advantages for automated visual inspection:

- **Traditional Techniques:**

  - **Accuracy in Localisation:** Traditional methods excel in post-processing localisation accuracy, ensuring precise identification of components within their designated regions.
  - **Component Awareness:** The system is primed to search for a specific component within a predefined area, improving efficiency.
  - **Challenges in Differentiation:** Despite its strengths, traditional inspection may struggle to distinguish between similarly shaped components, potentially leading to misidentification.

- **YOLO AI:**
  - **Class Differentiation:** YOLO AI's training enables it to distinguish between classes that appear similar, improving the identification of different components.
  - **Robust Classification:** The model can accurately classify components even when differences are subtle due to its deep learning architecture.

The integrated inspection process leverages both approaches effectively. As components are extracted from their predicted positions on the board, after coordinate extraction using conventional method, the whole board image is then passed to the YOLO model, which identifies the relevant classes.

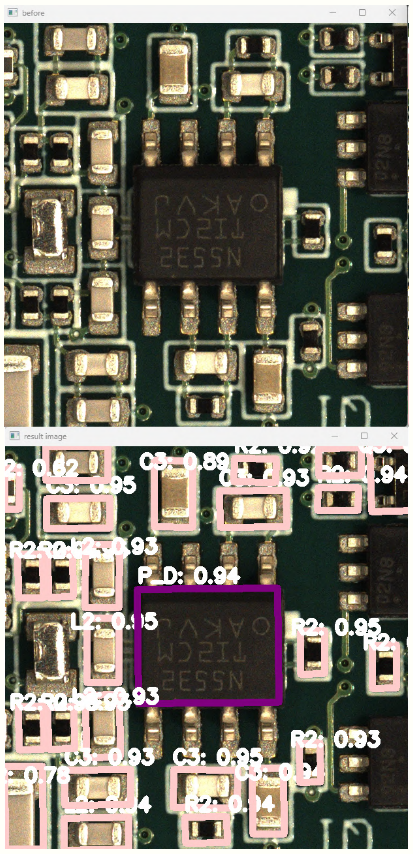


Figure 4.63: Power Distribution detection using AI and Traditional method

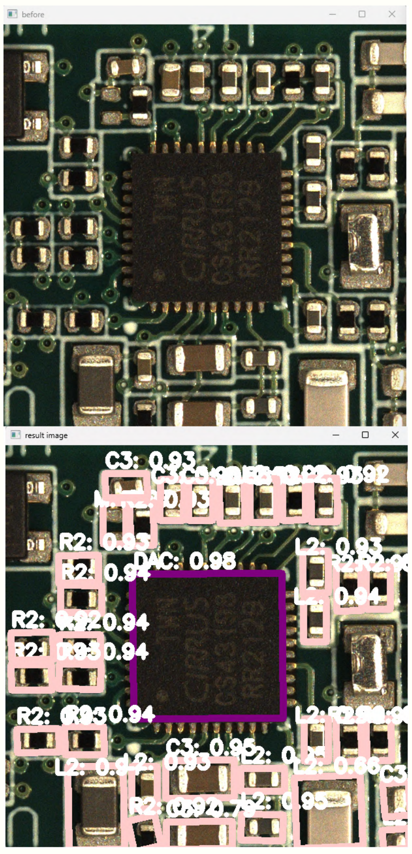


Figure 4.64: Digital to Analog Converter detection using AI and Traditional method

The images above illustrate the input and output of the YOLO model of the Combined inspection.

Inspection Process Flow

The inspection process follows a systematic approach, iterating through each component to be inspected, which was extracted from the board. The key stages of the workflow are outlined below:

1. **Components Coordinates Extraction:**
   - Before the start of the inspection, the program locates the reference points using template matching. These reference coordinates are verified by comparing the distances between the reference extracted from the template-matching technique with the distances already saved in the golden template.
   - When reference points are verified, the components' coordinates to be inspected are then extracted. The values extracted are based on the saved coordinates between the coordinates and reference points saved in the golden template.
   - Therefore, the program would have a list of component coordinates.
2. **YOLO-Based Detection:**
   - After the expected components' coordinates are extracted, the board image size is calculated, and the longer edge size is calculated. This longer edge is then passed to the YOLO model to dynamically process the image without resizing it.
   - The image of the board and its longer edge size is then passed to the model, and the output is stored in a variable called `self.foundComponents`
   - The output of the model is object classes and coordinates. When all the values are stored, the program starts iterating over the components to be inspected.
3. **Reset Failure Flags:**
   - At the beginning of each iteration, failure flags for the current component are reset to ensure accurate tracking.
4. **Missing Component Detection:**
   - The central section of the component is analysed using the `is_color_within_range` function to determine whether its colour matches the board background across multiple colour spaces.
   - If the background colour is detected, the `isMissing` flag is set
5. **Coordinate Verification:**
   - When the component passes the initial test, it goes to the next test, where the predicted coordinates are matched with the found result coordinates. On one hand, the found coordinates are the result of the YOLO model, while the predicted coordinates are the extracted coordinates calculated using the conventional template matching technique.

- A filtering logic is applied to find results that specifically overlap with the component being inspected predicted location:

```
for comp in filtered_components:
    diagonal_diff = abs( math.sqrt(comp["h"]**2+comp["w"]**2)
                        -math.sqrt(w_o**2+h_o**2))
                        /math.sqrt(w_o**2+h_o**2)

    if ( comp["x"]<xci+w_o/2 and
        comp["x"]>xci-w_o/2 and
        comp["y"]>yci-h_o/2 and
        comp["y"]<yci+h_o/2 and
        diagonal_diff < 0.08):

```

- **Class Name Mismatch:** If a match is found, but the class name does not match the expected value, the `wrongPart` flag is set.
- **No Matches Found:** If no matches are found, the component is marked as missing `isMissing`.
- **Match found:** If a match was found with the correct class name, the component image goes to the next inspection step.

6. **Template Matching and SIFT Algorithm:**

   - **Template Matching:** If a match is found which has the correct class name, template matching is used to verify the presence of the component.
     - If template matching succeeds, the component is confirmed as found. Otherwise, the program uses the secondary templates stored in `self.templateDict` and iterates through the whole catalogue until either a match is found or the program will use the SIFT Algorithm. As mentioned in the traditional method, the secondary template catalogue stores images of good components that can be used for inspection.
   - **SIFT Algorithm:**
     - If template matching fails, the SIFT algorithm is used to identify potential rotation:
       * If the component is present but rotated beyond the acceptable threshold, the wrong orientation flag is raised `wrongOrientation`.
       * If SIFT does not find the component, the `wrongPart` flag is set

Summary of the Traditional and AI-Based Inspection Techniques

The following summarises the inspection process that integrates both traditional and AI methodologies:

1. **Components' coordinate extraction:**

- The components' coordinates are extracted using template matching conventional technique and then stored for comparison with the YOLO result

2. **YOLO-result:**

   - After calculating the size of the image and passing the image and its size to the YOLO model, components' coordinates and class names are extracted and stored to be compared with the conventional technique results.

3. **Component Iteration:**

   - The iteration through the components to be inspected starts.

4. **Component Colour Identification:**

   - Use the board background colour space check to determine if the component is missing.
   - If the central section of the component image matches the board's background colour across all spaces, classify the component as missing.

5. **Comparing and Evaluating Results**:

   - The Program loops through the YOLO results and compares it with the inspected component's coordinates.
   - **Class Mismatch:** If matches are found but the identified class name differs from the expected component, it is classified as a wrong part depending on the logic output.
   - **No Match:** If there were no matches, classify the component as missing.
   - **Match Found and Class Match:** If a match was found

6. **Template Matching and SIFT Processing**:

   - **Template Matching**:
     - If the class name matches, perform template matching to confirm the component's presence.
   - **SIFT Algorithm**:
     - If template matching does not find a match, apply the SIFT algorithm to validate further.
     - If SIFT finds no match, classify the component as the wrong part.
     - If SIFT finds a match but with a rotation angle beyond the acceptable threshold, raise the 'wrongOrientation' flag.
     - If SIFT succeeds within the allowable angle, mark the component as correctly found.

7. **Conclusion**:

- Combining traditional techniques and AI models provides complementary strengths, ensuring a thorough inspection process:
  - **AI Alone**: While effective for identifying components despite size or board variations, it struggles with accurate localisation.
  - **Traditional Alone**: Although useful for precise localisation, it might mistakenly classify one component as another due to similar visual patterns.
- **Best Approach**: The combined use of traditional and AI techniques ensures accurate localisation and reliable classification, reducing false positives and negatives while delivering good inspection results.

4.5.7 Viewing the Board Post-Inspection

After the inspection process is complete, the user can visually examine the results through highlighted areas on the board image. These highlights categorise components into ”good” (without defects) and ”bad” (with detected issues), offering a clear indication of the inspection outcomes.

Good Components:

- Components that passed all the checks are highlighted in the board image, ensuring their positions align with expected coordinates.
- The following board image exemplifies a defect-free board with all components classified as correct:

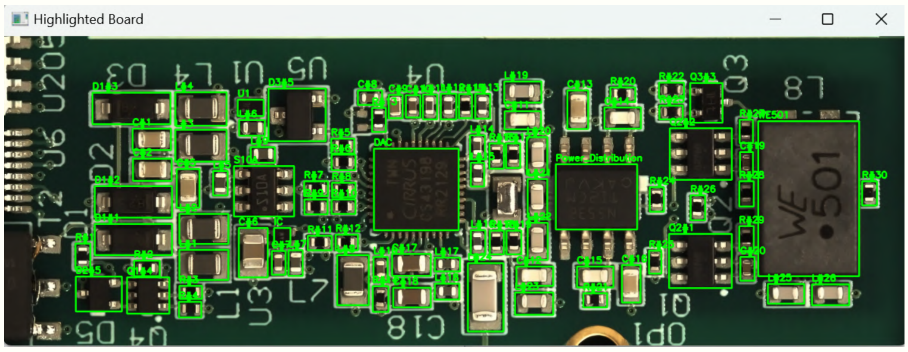


Figure 4.65: Post inspection board visualisation

Defective Components:

- Components flagged for issues like wrong part, wrong orientation, or missing status are distinctly marked to aid further investigation.
- The board image below displays a board with defective components, clearly differentiating between valid and erroneous parts:

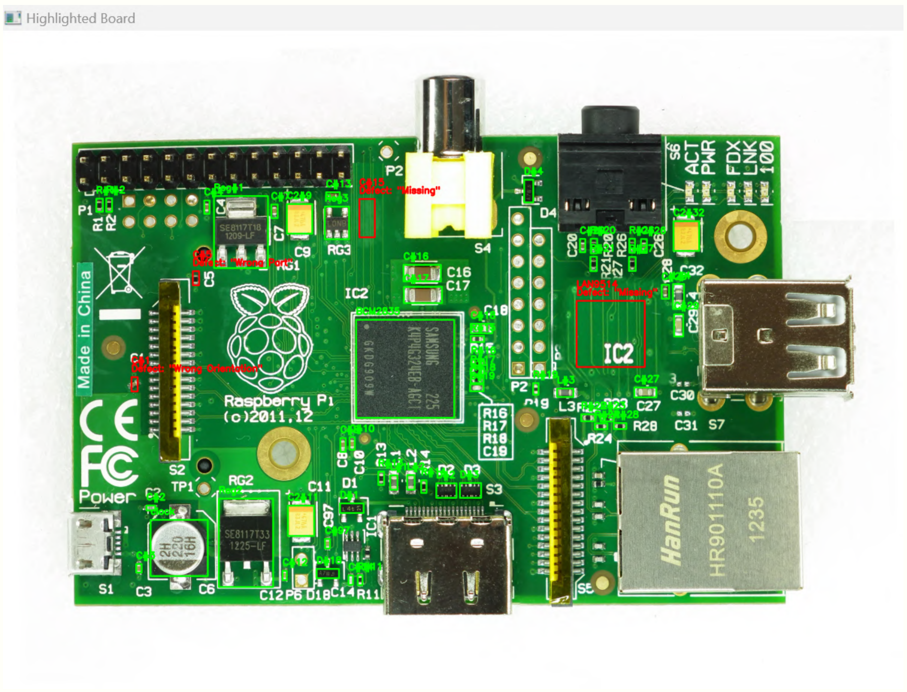


Figure 4.66: Board with defects

Chapter 5: Results and Discussion

5.1 Results and Observation

In the following sections, we will present the results of the board inspections, showcasing the time taken for each inspection method and image type used. We will also provide tables indicating the number of mismatches and the robustness of the inspection process using different techniques. In addition, we will discuss the effectiveness of the inspection system based on the dataset obtained.

This thesis had two software built—one for labelling and analysing the golden board to create a reference for the inspection process. The second software is for the inspection process itself.

The system was tested on 14 board images named boards A to N, which were mostly defect-free. Boards A to N are similar to the following figure, which is the golden board image chosen as a template for the first software to extract the golden component template.

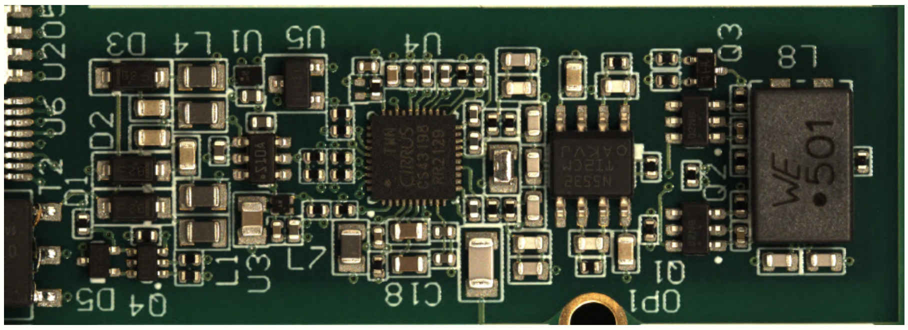


Figure 5.1: Golden Board

Three inspection methods were used: traditional, AI-based, and Combined Traditional and AI-based inspection. Each inspection method had four types of images to inspect: Coloured Images, Grey Images, Filtered Coloured Images, and Filtered Grey Images. This means that there are 168 results to analyse.

The traditional method took the component image, passed it through a template-matching process, and evaluated the integrity of the component. The second method of inspection powered by the YOLO AI model passed the full image of the board, classified the components, and returned their coordinates. The coordinates were then verified and matched with the template component coordinates, and when the location was confirmed, the integrity of the component was evaluated. As for the third method, which utilises both techniques, the component location was predicted first using the traditional method of localisation, and then the board image was passed to the AI model to classify the components in it. After classifying the components, the result goes through a filtering process to eliminate extra classified components in the image and to find components that match the class and coordinates of the predicted coordinates stored at the beginning of the process. Then, the component integrity is evaluated.

Two YOLO AI models were used for inspection, each trained using a different method. The first method was to pass the full board image to the AI model and adjust the hyperparameter to allow learning from the full image without data loss. On the other hand, The second model was trained by taking the dataset images and dividing them into smaller sub-images. By that, the second model would expect smaller images when used. Both provided satisfactory results. However, the first method of training had a better advantage due to the larger and defect-augmented dataset.

Integrating traditional and YOLO AI-based techniques provides a comprehensive approach to PCB inspection, balancing the strengths and weaknesses of both methods. The traditional methods offer precise localisation but struggle with components with similar visual patterns. On the other hand, the YOLO AI methods excel in identifying components across varying sizes and board variations but may face challenges in accurate localisation.

Combining these techniques ensures thorough inspection, reducing false positives and negatives and improving overall inspection accuracy. The tables presented highlight the time efficiency, mismatch rates, and system robustness, demonstrating its practical application and reliability in real-world scenarios.

5.1.1 Inspection Time Analysis

The graphs below made from table A.1 show the inspection time for each board depending on the inspection method and image type used. Each board is divided into three columns representing the inspection methods: “Traditional”, “YOLO”, and “Combined” across four image types (Coloured, Grey, Filtered Coloured, and Filtered Grey).

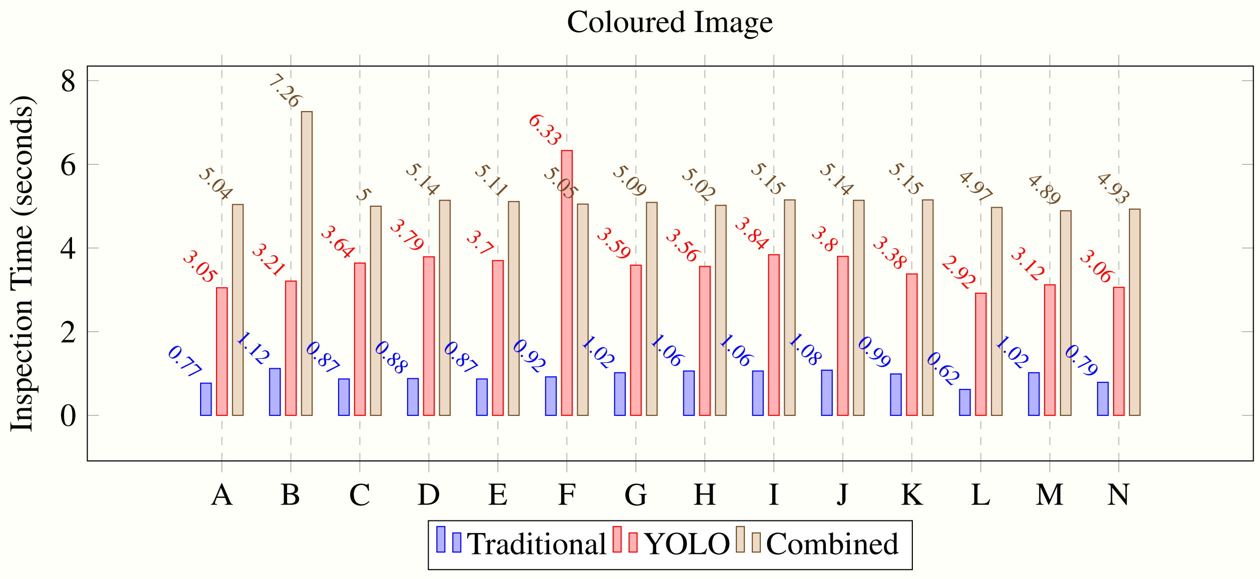


Figure 5.2: Inspection Time Analysis - Coloured Image

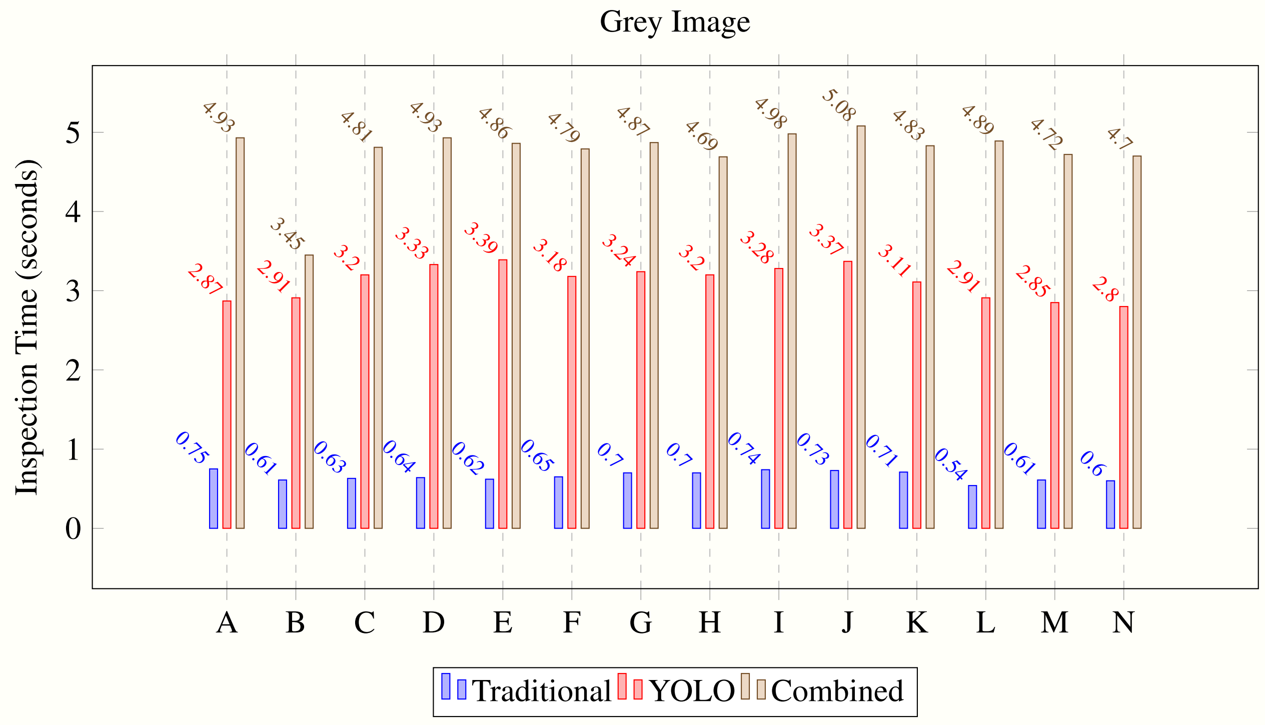


Figure 5.3: Inspection Time Analysis - Grey Image

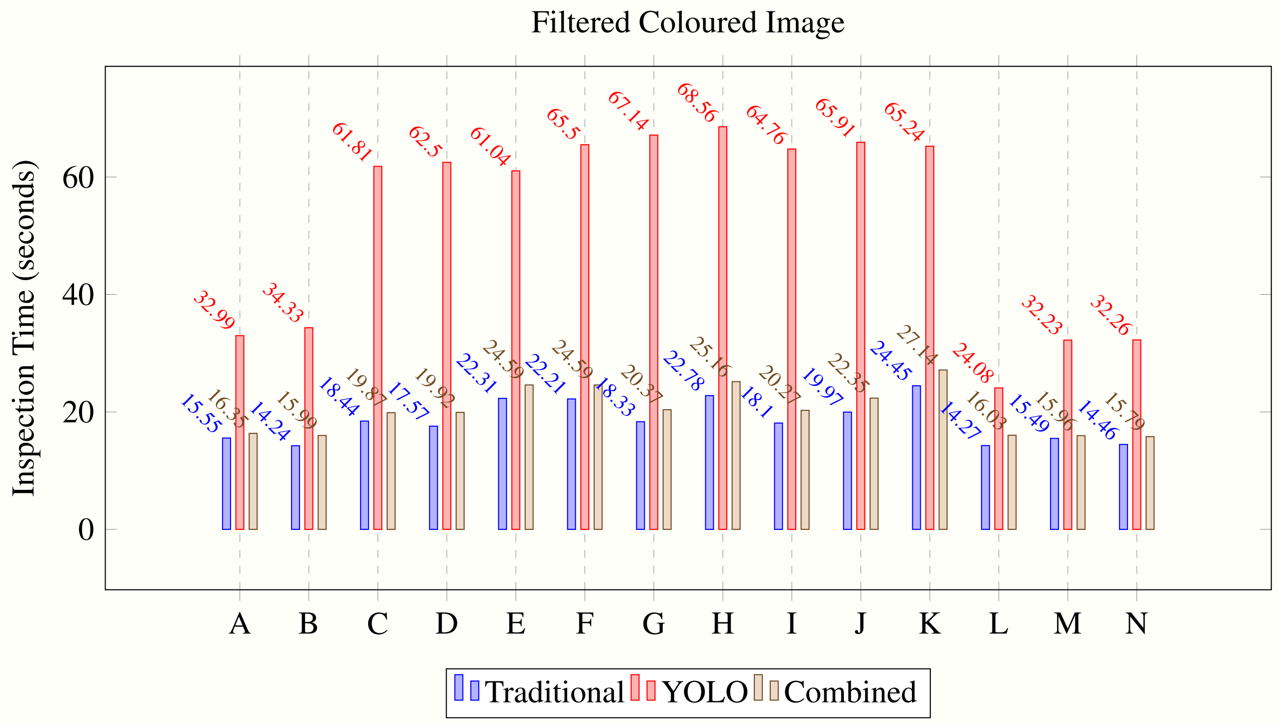


Figure 5.4: Inspection Time Analysis - Filtered Coloured Image

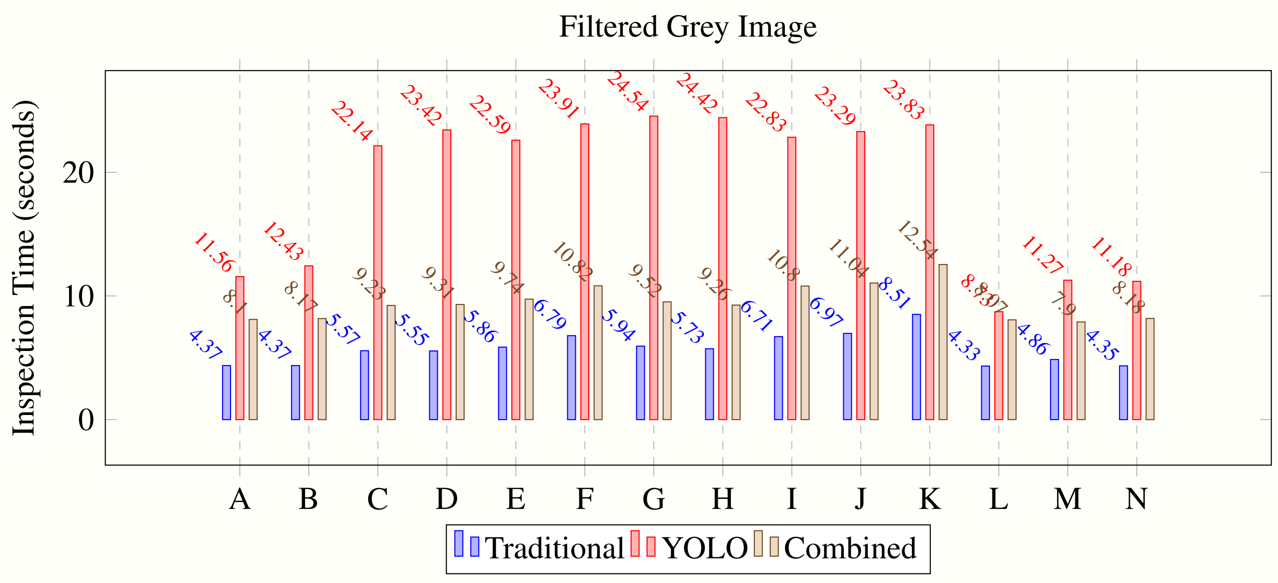


Figure 5.5: Inspection Time Analysis - Filtered Grey Image

The traditional method consistently shows the shortest inspection times, particularly for unfiltered images, with times ranging from 0.77 to 1.12 seconds.

In contrast, leveraging deep learning, the YOLO method exhibits longer inspection times, especially for filtered images, ranging from 24.08 to 68.56 seconds. That is due to the process itself; when the program determines a component is defective, more time is taken to confirm that this is true. The time increase is due to the nature of the YOLO AI inspection process as the program loops through all found coordinates from the YOLO AI result to match it with the position being inspected and breaks when a match is found. Therefore, the inspection time would ideally be around 24 to 33 seconds for the filtered images. The defect found in the boards during the inspection process will be explained in the next section.

The combined method results performed well, as the inspection times ranged from 15.79 to 27.14 seconds when using coloured filtered images. Ideally, in all image types, there would be a slight increase in the inspection time over the other process. That is due to the overhead of sequentially processing the board. As the board has already determined the location of the components, it confirms these locations with the output coordinates of YOLO AI.

5.1.2 Mismatch Analysis

The graphs below made from the following table A.2 show the number of times mismatches occurred during the inspection process and 4 image types, indicating False True and False False outcomes.

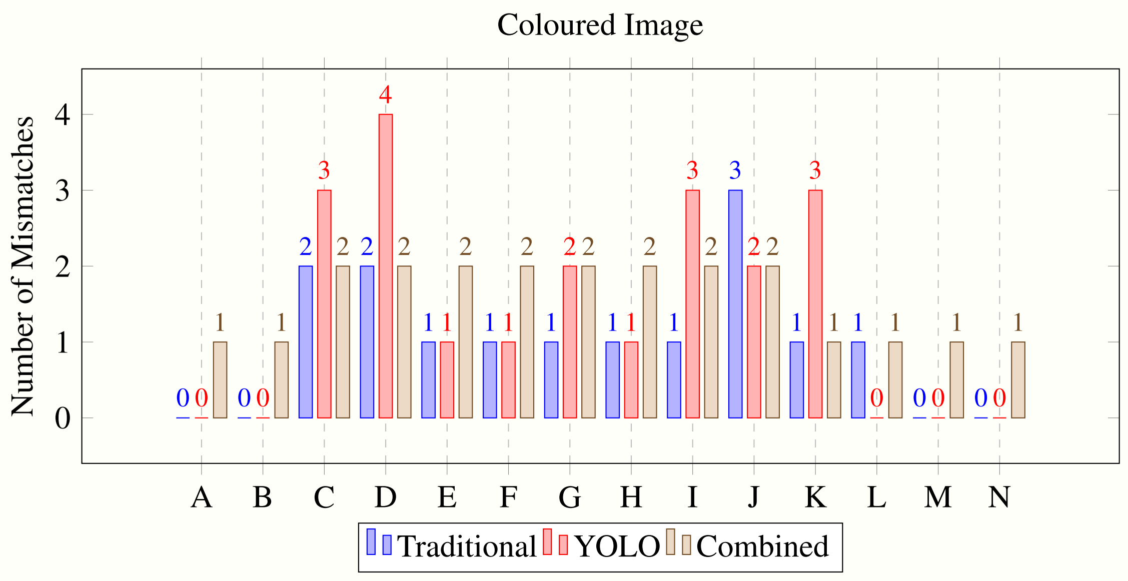


Figure 5.6: Mismatch Analysis - Coloured Image

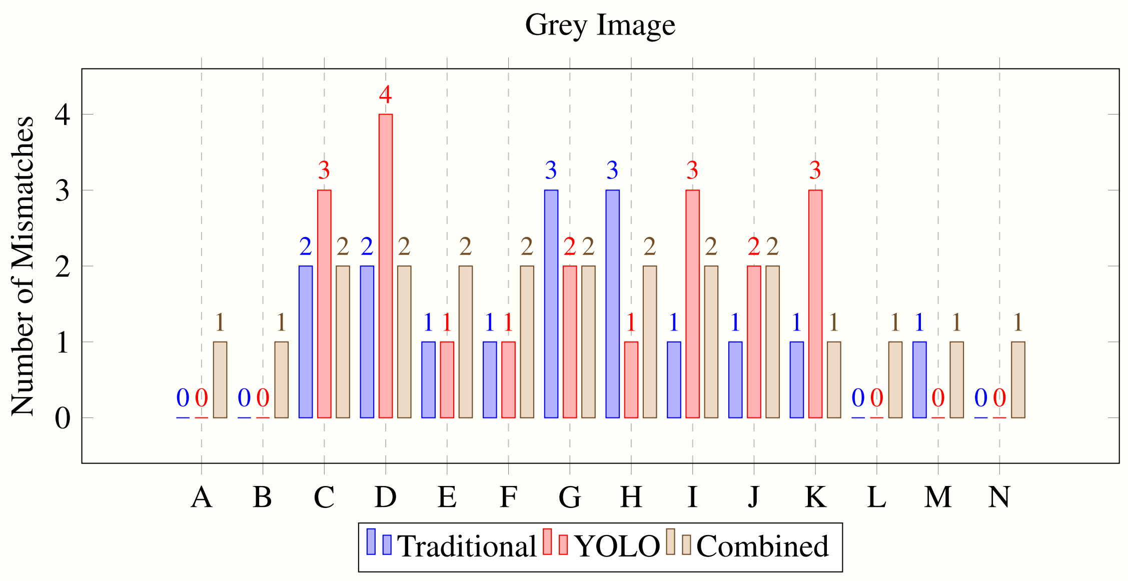


Figure 5.7: Mismatch Analysis - Grey Image

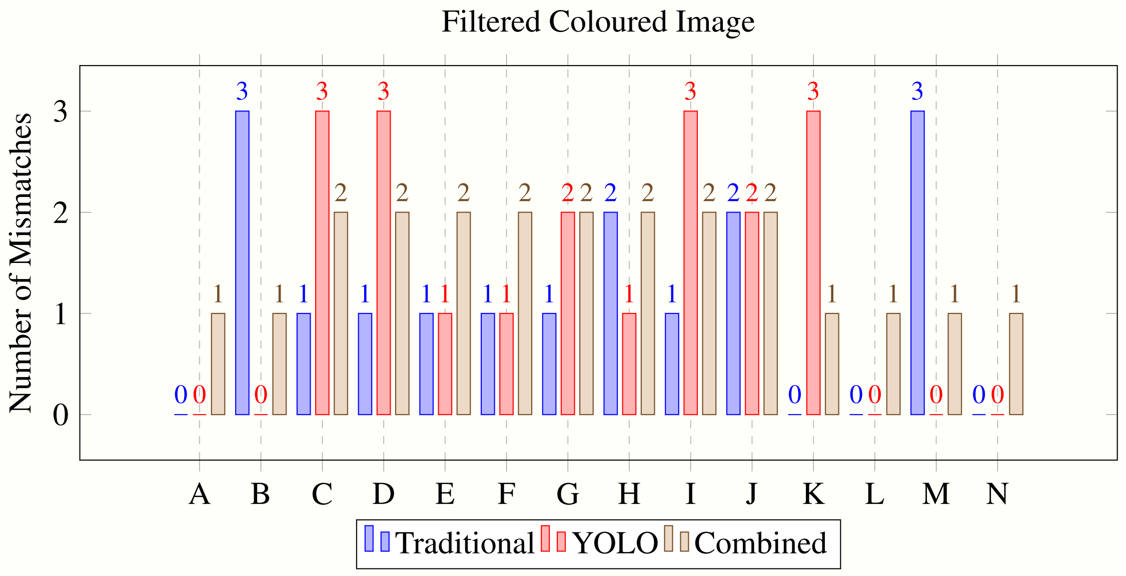


Figure 5.8: Mismatch Analysis - Filtered Coloured Image

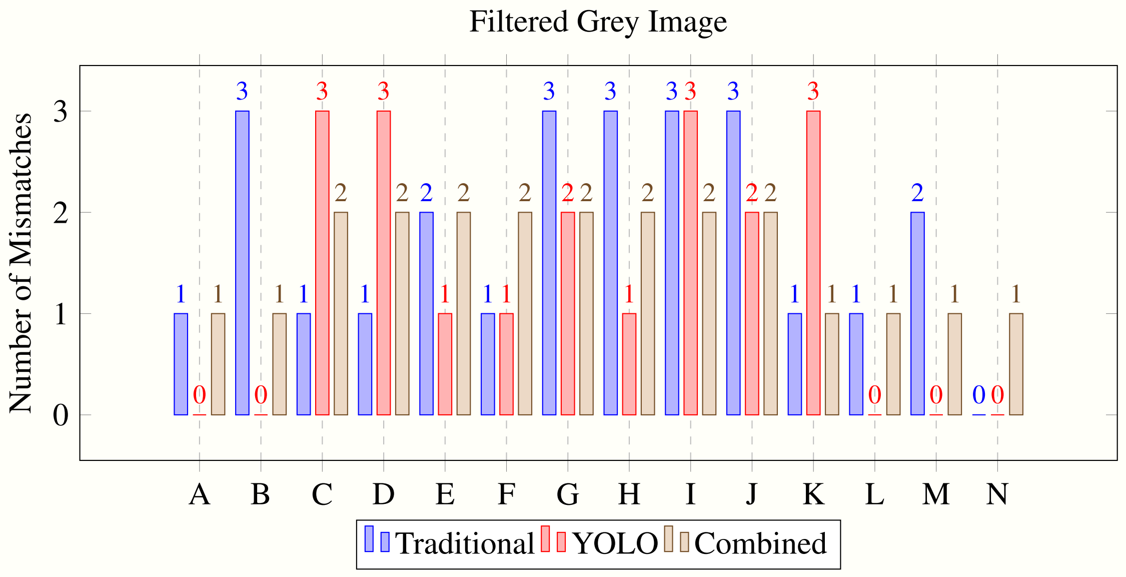


Figure 5.9: Mismatch Analysis - Filtered Grey Image

From the results, it is evident that the combined ”Traditional and YOLO” method regularly achieves 1 to 2 mismatches, only False False, highlighting its reliability. One of the mismatches that happened was due to a human labelling error on some boards. The other mismatch that occurs is due to not enough datasets for training the board. The second mismatch happens with the ”WE501” component due to its large size and the nature of the size of the training image passed to the second method of training in the YOLO AI model; this component was not always available in the images in addition, it was not in practice to label a part of the component and not the whole component, which resulted in low detection rate. The only way this component would be detected is if the component was centred in the middle of a 640*640 image.

On the other hand, the YOLO method, which used the AI model produced from the first training method, performed better, although there were more mismatches. Ideally, the first method of inspection would not produce any error. The reason this happened is that the first training method lacked a variety of BMS boards, although they were augmented with a big dataset. On the other hand, the second training method, although smaller, included a variety of boards of the same type to train the model. This would prevent any mismatches from occurring in the YOLO model, unlike the first training method. However, what is not shown here is that the first model was trained on a dataset with augmented defects like missing or misplaced components. Unlike the second model, the first AI model was able to successfully classify the components that were displaced from their original position, proving that the first training method was better.

Lastly, the traditional method virtually produced a comparable amount of mismatches to the other techniques. However, in essence, using template matching without the use of the secondary templates stored in `self.templateDict` this technique would produce a lot of mismatches.

Unlike the mismatches produced in the YOLO AI inspection method and the combined inspection method, the mismatches produced with the traditional techniques are unavoidable. One of the mismatches occurring with the traditional technique is when the orientation marker of a component is near the centre of the component; the program might calculate the orientation of the component wrong.

In addition, template matching and sift techniques used in this inspection method focus more on the shape of the component than its colour values. Therefore, sometimes False True results are produced when components have similar shapes but are actually different colours.
To clarify, in template matching, the method used is `cv2.TM_CCOEFF_NORMED`, which means the template matching does not inherently focus on colour information. Template matching primarily depends on the structure and pattern of pixel intensities. If the shapes of the components are similar, the pixel patterns will match despite differences in colour. Normalisation methods further reduce sensitivity to colour differences by focusing on structural correspondences.
On the other hand, the SIFT algorithm compares two images that are actually in grayscale, which means it disregards colour information. The primary focus of the algorithm is on the shape and structure of the components. SIFT detects key points in an image and describes the local appearance around each key point using the gradients. The key points are identified based on the structure and intensity changes, such as edges and corners.

5.1.3 Frequency of Golden Template Failure

The graphs below made from the information from table A.3 show the number of times the inspection failed to use the golden template to identify the inspected component and had to use the additional template dictionary images of components of the same type to match successfully. This indicates the robustness of the golden images and the effectiveness of the ever-learning software.

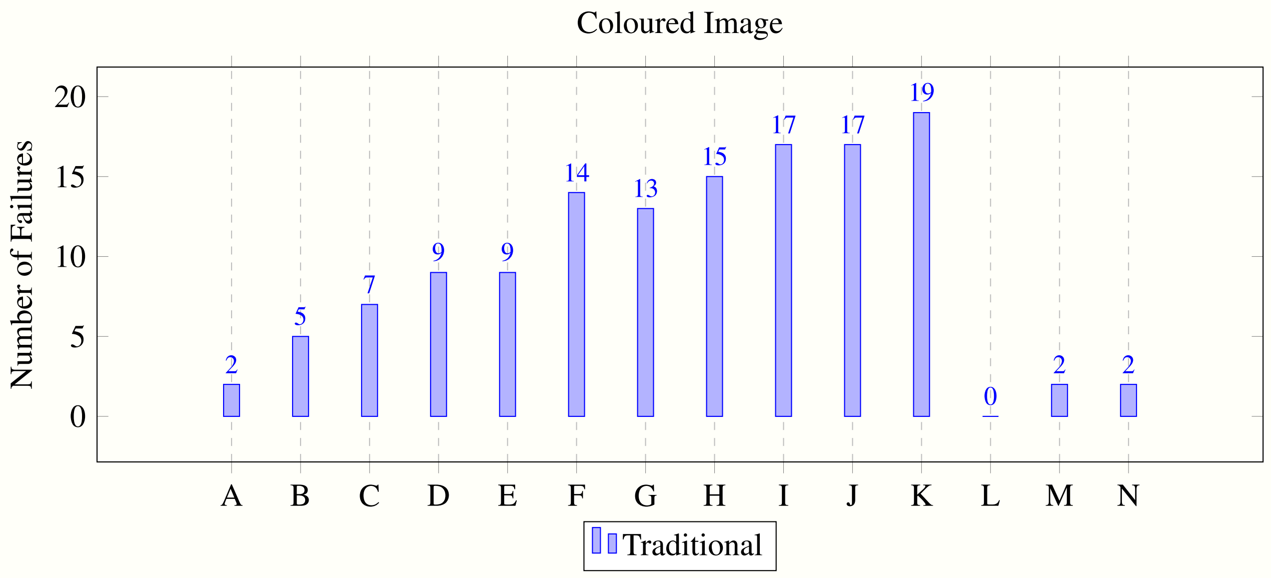


Figure 5.10: Template Failure Analysis - Coloured Image

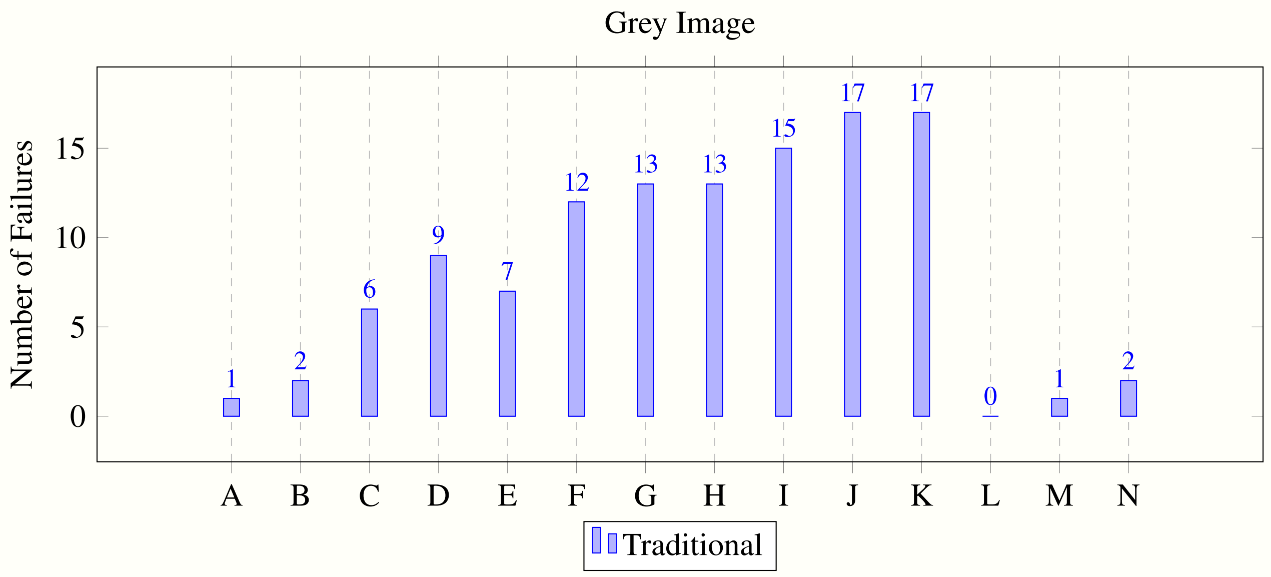


Figure 5.11: Template Failure Analysis - Grey Image

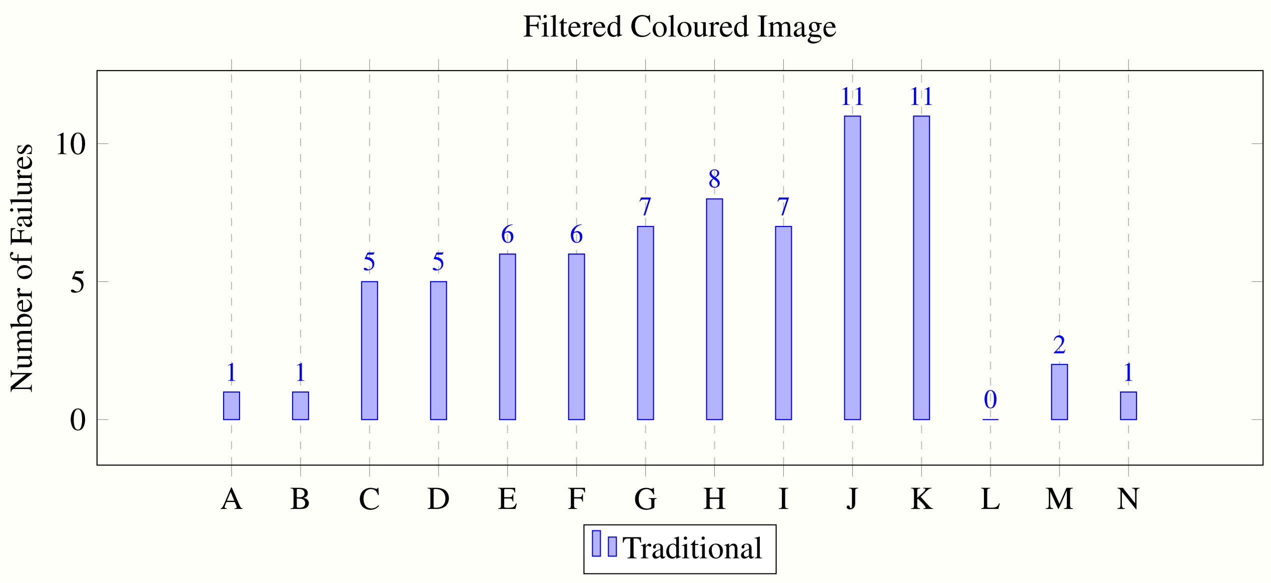


Figure 5.12: Template Failure Analysis - Filtered Coloured Image

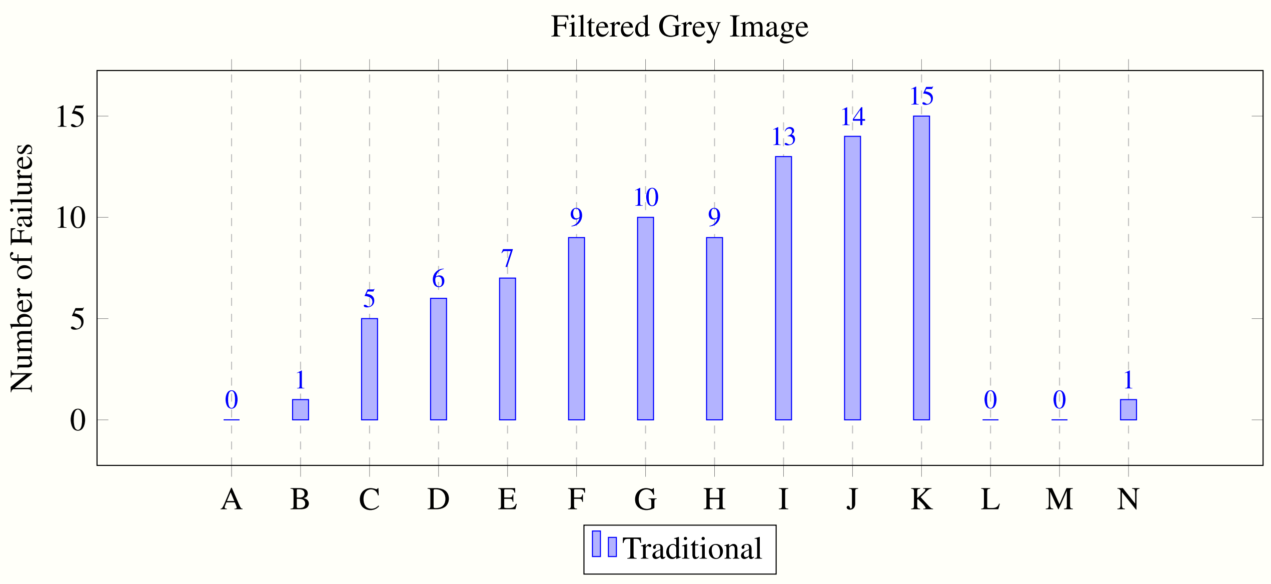


Figure 5.13: Template Failure Analysis - Filtered Grey Image

As shown from the graphs, due to the high sensitivity of the template matching technique to change in the traditional method, the program had to use the secondary template from the additional template dictionary in most boards. It is also worth mentioning that filtering the image before passing it to the template-matching process reduces the use of a secondary template dictionary.

5.1.4 Traditional Inspection Technique

The traditional inspection technique has certain sensitivities and limitations. The following points summarise the findings and improvements made to enhance the traditional inspection system:

- **Sensitivity to Change**:
  - The traditional inspection system was highly sensitive to even minor changes.
  - This sensitivity often resulted in numerous False False results, where defect-free components were incorrectly flagged as defective.
  - To address this issue, a feedback feature was implemented, allowing the system to learn from its mistakes. New template images, which were initially unrecognised by the golden template, were fed back into the system, creating a comprehensive template dictionary with varied images of the same component type, which was saved in the following dictionary `self.templateDict`.
- **Challenges with Simulated Defects**:
  - When tested on a dataset with simulated defects (since the original dataset was nearly defect-free), the traditional technique produced several False True results.
  - This was primarily because some components had similar shapes, sense template matching `cv2.TM_CCOEFF_NORMED` method used, and the SIFT algorithm inherently focuses on the component structure and shape, leading to incorrect identifications.
- **Strengths in Localisation**:
  - Despite the above challenges, the traditional technique proved highly effective in localising component coordinates on the PCB.
  - The margin of error in localisation was almost nonexistent, demonstrating the technique's precision in identifying exact component positions.

To summarise, while the traditional inspection technique has notable strengths, particularly in precise localisation, its sensitivity to changes and challenges with similar component shapes highlights the need for supplementary methods to improve accuracy and reduce false results. Incorporating a feedback mechanism and expanding the template dictionary has significantly enhanced the system's reliability. However, if the board had many components replaced by another, the traditional inspection technique would frequently fail, producing many False Positive results due to the nature of the template matching and SIFT algorithm, which focuses on the component structure, which is unsatisfactory.

5.1.5 YOLO-based Inspection Technique

The YOLO AI-based inspection technique offers significant advancements in component classification but also presents several challenges. The following points summarise the findings and improvements made to enhance the YOLO-based inspection system:

- **Effectiveness in Component Classification**:
  - YOLO proved highly effective in classifying components.
  - The model's ability to classify components accurately is a significant advantage.
- **Necessity of Overcoming Localisation Issues**:
  - Overcoming the localisation of components is an essential step to improve the inspection process.
  - While the YOLO model classified components effectively, it was necessary to verify the component location to verify the classification accurately.
  - Components of the same type but different sizes posed a challenge, requiring post-processing calculations to confirm the dimensions and ensure the correct component identification.
  - **Post-Processing for Accurate Localisation**:
    - ∗ Even after identifying components, the exact location of the PCB component was not always clear.
    - ∗ This issue could lead to false localisation of components if the algorithms were not refined correctly.
    - ∗ There was a chance the components with close results could be mixed with other components, resulting in False results, but that was overcome by fine-tuning the localisation tolerances.
    - ∗ Implementing effective post-processing steps to calculate and verify component dimensions is crucial to overcome this challenge.

To summarise, The second method of using the YOLO AI model was perfect for classification. However, it requires fine-tuning for tolerances to verify the components' coordinates correctly. Which might lead to the need for fine-tuning for new boards. However, the results are very satisfactory.

5.1.6 Traditional Plus YOLO Inspection Technique

A combined approach was implemented in this inspection method to leverage the strengths of both traditional and YOLO techniques. The following points summarise the findings and improvements made to enhance the Traditional Plus YOLO inspection system:

- **Combining Strengths**:

  - The traditional technique's excellent localisation capabilities were combined with the YOLO AI's accurate classification abilities.

- **Accurate Localisation and Classification**:
  - The combined approach resulted in the accurate determination of component coordinates.
  - Components were classified correctly, enhancing the overall inspection process.
  - By integrating the template data frame, the technique demonstrated significant robustness.

- **Reduction in False Results**:
  - The hybrid method effectively reduced both False False and False True results.
  - This reduction indicates improved reliability and accuracy in the inspection process.

- **Human Error**:
  - Due to the mislabelling of one component during the training process, this method resulted in false true results in some boards. But that does not mean that the method itself is bad.

5.1.7 Component Tolerance and Thresholds in Inspection

In all inspection techniques, the threshold for acceptable components—whether it pertains to the template matching result or component size tolerance—was persistently tested and adjusted until the most acceptable tolerance levels were identified by the program logic. The following points summarise the key observations and adjustments made to optimise the inspection process:

- **Persistent Testing of Tolerances**:
  - Thresholds for template matching results and component size tolerance were continuously evaluated.
  - The program logic was fine-tuned to determine the most acceptable tolerance levels, ensuring accuracy in component inspection.

- **Impact of Larger Tolerance on Component Shift**:
  - Increasing the tolerance for component shift resulted in conflicts when matching components to their true locations.
  - This adjustment led to higher rates of incorrect localisation, highlighting the need for precise tolerance settings.

- **Observation on Orientation Markers**:
  - Components with orientation markers positioned close to the centre were more prone to False False results.
  - This finding indicates that the location of orientation markers significantly impacts the accuracy of component identification.

Chapter 6: Conclusion and Outlook

6.1 Summary

This thesis explored the implementation of Traditional techniques and CNN in PCBs' automated visual inspection process. The primary objective was to enhance the accuracy and efficiency of PCB inspections while eliminating the reliance on manual inspection methods.

This thesis introduces three inspection methods: Traditional, AI-Based, and Combined Traditional and AI-based inspection methods. In addition, Two AI models were used in the inspection process. The first AI YOLO model was used in the AI-based technique, and the second AI YOLO model was used in the Combined technique.

The first Inspection method initially predicts component location by finding the three points of reference and using trilateration. After extracting the component images, the inspection uses conventional template matching and the SIFT algorithm to ensure component integrity.
The second method utilises the first YOLO AI model to identify components by passing the full image to the model. The YOLO model result has the component's class and coordinates. The results go through post-processing to match the component's class and location with the golden template, ensuring component integrity.
The third combines both traditional techniques and AI methods and uses the second AI YOLO model in the inspection process. As a pre-processing, the third method first predicts the components' locations by finding the three points of reference and then uses them as anchors to Localise the components using trilateration. After extracting the location of each component, the full board image is then passed to the AI model, which gives back coordinates and the class of each component in the image. Then, the program iterates through each component to be located and compares the predicted coordinates extracted from the conventional techniques with the coordinates extracted from the YOLO model matching the coordinates. After that, the program extracts the component image of the successfully matched pairs of coordinates. This image goes through template matching and the SIFT matching process for additional verification and confirmation of the correct orientation of the component.
The third technique demonstrated better performance, leveraging the precise localisation capabilities of the traditional method and the near-perfect classification accuracy of the YOLO AI model. However, the primary trade-off for the third method was the increased processing time compared to the first and second methods

Two methods were employed to train the AI model. The first method involved using a full image

of the PCB. The dataset of the first model was made from 2 images of 2 different board types - a board provided by BMS-Elektronik and a Raspberry PI board- from these 2 images, a total of 3689 images of simulated defects of the BMS-Elektronik board and 1659 images of simulated defects of the Raspberry PI were created, a total of 5348 boards images to train the AI model.
The second method involved taking 14 board images provided by BMS-Elektronik and dividing them into sub-images of 640x640 pixels, resulting in a total of 546 images with no simulated defects. These 546 divided images were passed to the AI model to train the AI model.

The first method yielded promising results; however, the down-scaling of the image led to data loss of small objects in the images, resulting in the non-detection of some critical objects necessary for the localisation process, a crucial part of the inspection. After adjusting the hyperparameters and training the model to expect images of sizes 2048x2048 pixels, the model successfully detected all components and performed efficiently. The first model had a mAP of 99.5% and worked quite efficiently on test images with satisfactory results.

Consequently, the second method of training the AI model was built and engineered to accommodate and expect smaller images, where it takes each component and extracts its image with its surroundings to avoid the inconsistency caused by different image dimensions. The second method achieved a mAP of 99.5%. However, dividing a full board image into sub-images and passing them to the AI model took a lot of time, around 60 seconds for a board with 89 components. Therefore, the third inspection method was modified to accommodate the full image of the board instead of sub-images; the model was used with a full image of the board in the third inspection method; although it is bigger than the training images, YOLO AI can support this feature by dynamically passing the size of the image to be inspected to detect the components. Which worked perfectly except with component ”WE501”, as it was the largest component. This means that the number of images that included this component in training was less compared to the other components. The only method to detect this component is by cutting the component out of the full image and passing the component alone to the AI model.
when both models were used to test new unseen PCB images, they produced satisfactory results with a prediction confidence level of 0.9, where most components were classified correctly, and other untrained components were not classified as expected.

6.2 Conclusion

As mentioned in the previous section, three methods of inspection were created for this thesis, and two AI models were embedded in the second and third methods. In this section, we will go over the conclusions drawn from each method, as well as the objective questions mentioned above.

The first method of inspection was sensitive to change and resulted in many mismatches initially due to the nature of using template matching. Which would total to around 118 mismatches out of 1246 inspected components, a score of 9.47%. The mismatches were reduced using the SIFT algorithm. However, it was still not satisfactory. Therefore, a feedback loop mechanism was developed to use as a secondary template reference that matches the target component with new template images stored as a backup reference, which grows when a new False False result occurs, which reduced the mismatches in the traditional technique to 14 mismatches with a 1.12% failure percentage. This failure is due to other factors of integrity verification, such as a mismatch of writing on the component or the rotational marker on the component orientation calculation error. Which might seem like a good result. However, when the traditional method was tested on the boards with simulated defects, it resulted in False True results, especially when the components were replaced with other components with similar shapes due to the mechanism used in inspection as template matching and sift algorithm focuses on the structure of the object and are not affected by colour change of the components. This result is not satisfactory.

The second method of inspection using the YOLO AI model gave satisfactory results where it was able to classify components on the 14 BMS-Elektronik boards effectively, although it was trained on 1 variation of this board, which was augmented with simulated defects. Although it seems that the 2nd method produced more mismatches with the 14 test board imager, these mismatches were due to the limited amount of board variation used in training this model. If the model had been trained with more boards, zero mismatches would have resulted. Another point worth mentioning is that when the method was tested on a board with simulated defects, it classified the components correctly and labelled their defect type correctly. The only drawback of this method is the need to fine-tune the tolerances for localisation, which is very critical in the inspection process. If the tolerances have the wrong value, it can produce many False False results, as the model outputs might get mixed up with each other. It is hypothesised that if a new board type needs to be inspected, there is a high probability that the backend of the inspection software will need to be adjusted to accommodate the new tolerances. However, the second method is quite satisfactory.

As for the Third method, it utilised the strengths of both previous methods. Therefore, fine-tuning was not required to adjust component localisation tolerances, and it would rarely classify components falsely. The third method had almost zero mismatches. However, due to a labelling error during AI training, the method did not reach zero mismatches.

When training the AI model, it is crucial to maintain consistent image dimensions, as the size of the images must remain fixed. To ensure the most optimum results, the images used during prediction should match the dimensions and scale of those used in training. This consistency is key to the model's performance. Otherwise, if there is a very big image in the dataset compared to the other images, there might be data loss because the training mode tends to scale down the image during training, which would affect the prediction mode when trying to extract results during the inspection.

It is possible to pass the size expected of the image during prediction if the new unseen image has a bigger or smaller size; however, maintaining a similar size of training and prediction was found to be most optimal and satisfactory.

YOLO-OBB was selected for its ability to determine the orientation of components. However, due to occasional unreliability in the angles produced by YOLO-OBB, it was deemed preferable to utilise template matching to ensure accurate angle determination.

The objective questions raised in the introduction explored whether CNN could enhance the PCB inspection process. The findings indicate that using YOLO to power the inspection process with CNN significantly improved the classification of components, addressing the high sensitivity of traditional methods to changes. However, there were some limitations associated with using YOLO AI. Post-processing the AI results proved challenging, as accurate localisation was necessary after initial processing. Additionally, some small components required for the trilateration method were not detected initially, but this issue was resolved by optimising the training technique. Another challenge was the lack of a large dataset, which was mitigated by simulating defects and altering component locations to create more variation.

In addition to addressing the objective questions, several other conclusions were drawn. Integrating traditional techniques with AI technology has proven to be a robust method for PCB inspection, effectively combining the strengths of both approaches: traditional techniques for localisation and component orientation verification and YOLO AI for component classification.

Another key conclusion is the importance of maintaining consistent conditions when capturing board images for inspection, as even slight variations can lead to false results, such as misclassifications of components with similar shapes due to lighting changes.

To summarise the main points in the previous part, we conclude that the traditional method for inspection was not satisfactory due to the False True results. On the other hand, the second method of using YOLO AI produced very satisfactory results but would produce challenges in fine-tuning for component localisation. The third method utilised the strengths of both methods and produced very good results. We conclude that the second method would produce satisfactory results if the localisation fine-tuning tolerances challenges were overcome. The form of AI model training provided very satisfactory results; however, from the training process, we concluded several important points: The lack of a large dataset was a challenge we faced during training, which we overcame by creating a simulated dataset. Additionally, maintaining a constant size of images for training and passing the test images to the model proved to be the most efficient. We conclude that integrating CNN AI models in the PCB inspection process is necessary and would enhance the inspection process. The final statement is that the first method of AI training is the best method of training AI by capturing the whole board with its surrounding environment and simulating defects is the best. In addition, the second and third inspection methods are the best to use for inspecting PCBs.

6.3 Future Suggestions

Based on the findings of this project, several future improvements can be suggested to enhance the accuracy and reliability of the PCB inspection system:

Introducing a fourth reference point to the inspection system will significantly improve the accuracy of component localisation. Adding this additional reference point enables better localisation and verification of component positions, thereby reducing the likelihood of errors. This enhancement is hypothesised to be crucial for achieving more precise and reliable results in PCB inspections, which would lead to the perfection of the second inspection method, which uses the YOLO AI model only.

Another recommendation is to develop a dataset featuring images captured at various zoom levels. This approach aims to enable the YOLO AI model to identify components regardless of their scale and size variations. The AI can potentially improve its adaptability and accuracy in detecting components of varying scales by training the model with a dataset that includes different zoom levels. However, it is hypothesised that this strategy might yield negative results due to the increased complexity and potential inconsistencies in the training data. Despite this, the idea warrants exploration and testing to validate its effectiveness and determine if it can contribute to the robustness of the inspection system.

The final recommendation is for the method of inspecting components using template matching. To make the template matching sensitive to colours, we hypothesise that if the image was taken and separated into the three LAB or RGB colour channels, it might achieve colour-sensitive template matching. The method can be done by applying template matching on each colour channel separately and then combining the results. This method was applied quickly and experimented with in this work. It showed promising results; however, it needed fine-tuning. Therefore, it was preferred that would be added as a future recommendation.

Matching using LAB channels

```python
# Split the Lab channels
inspect_l, inspect_a, inspect_b = cv2.split(inspect_component_lab)
template_l, template_a, template_b = cv2.split(template_lab)

# Perform template matching on the a and b channels
res_a = cv2.matchTemplate(inspect_a, template_a, cv2.TM_CCOEFF_NORMED)
res_b = cv2.matchTemplate(inspect_b, template_b, cv2.TM_CCOEFF_NORMED)

# Combine the results
res_combined = (res_a + res_b) / 2

# Find the best match
_, max_val, _, max_loc = cv2.minMaxLoc(res_combined)
```

Matching using RGB channels

```
# Split the images into BGR channels
inspect_b, inspect_g, inspect_r = cv2.split(inspect_component)
template_b, template_g, template_r = cv2.split(template)

# Perform template matching on each channel
res_b = cv2.matchTemplate(inspect_b, template_b, cv2.TM_CCOEFF_NORMED)
res_g = cv2.matchTemplate(inspect_g, template_g, cv2.TM_CCOEFF_NORMED)
res_r = cv2.matchTemplate(inspect_r, template_r, cv2.TM_CCOEFF_NORMED)

# Combine the results (you can use different strategies, e.g., averaging,
    summing, etc.)
res_combined = (res_b + res_g + res_r) / 3

# Find the best match
_, max_val, _, max_loc = cv2.minMaxLoc(res_combined)
```

Implementing and testing these recommendations will help further refine the inspection process, ensuring higher precision and reliability in PCB inspections.

Academic References

[1] Khushi Singh, Shubhangi Kharche, Aakash Chauhan, and Pranali Salvi. Pcb defect detection methods: A review of existing methods and potential enhancements. volume 17, Nerul, India, December 2023. [Accessed on: 24-May-2024]. https://www.researchgate.net/publication/379270048_PCB_Defect_Detection_Methods_A_Review_of_Existing_Methods_and_Potential_Enhancements.

[2] C. Wu. Research on design and application of brand vision inspection and sorting system based on image processing. In *2020 2nd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*, pages 568–571, Taiyuan, China, October 2020. IEEE. Conference Date: 23-25 October 2020, Date Added to IEEE Xplore: 26 February 2021 [Accessed on: 24-May-2024]. https://ieeexplore.ieee.org/document/9360968.

[3] Abd Al Rahman M. Abu Ebayyeh and Alireza Mousavi. A review and analysis of automatic optical inspection and quality monitoring methods in electronics industry. volume 8, pages 183192–183271, 2020. [Accessed on: 24-May-2024]. https://ieeexplore.ieee.org/document/9214824.

[4] C. Neubauer. Intelligent x-ray inspection for quality control of solder joints. volume 20, pages 111–120, 1997. [Accessed on: 24-May-2024]. https://ieeexplore.ieee.org/document/622881.

[5] Harm van Schaaijk, Martien Spierings, and Erik Jan Marinissen. Automatic generation of in-circuit tests for board assembly defects. In *2018 IEEE International Test Conference in Asia (ITC-Asia)*, pages 13–18, 2018. [Accessed on: 24-May-2024] https://ieeexplore.ieee.org/document/8462941.

[6] Cal Houdek. Inspection and testing methods for pcbs: An overview. White paper, Caltronics Design & Assembly, Inc., 2023. [Accessed on: 24-May-2024] https://static1.squarespace.com/static/62e3c6284ddd8f7896c21906/t/643ea484dd7ee32ad15a78b7/1681826950765/CD%26A+White+Paper+%23401.pdf.

[7] Wei Chen, Zhongtian Huang, Qian Mu, and Yi Sun. Pcb defect detection method based on transformer-yolo. volume 10, pages 129480–129489, 2022. [Accessed on: 26-May-2024], https://ieeexplore.ieee.org/document/9978605.

[8] Shams Rehman, Ka Fei Thang, and Nai Shyan Lai. Automated pcb identification and defect-detection system (apids). volume 9, page 297, 02 2019. [Accessed on: 19-July-2024] https://www.researchgate.net/publication/332545825_Automated_PCB_identification_and_defect-detection_system_APIDS/citations.

[9] Abdel-Aziz, Fathi E. Hassanin, Ghada M. Abd El-Samie, and El Banby. A real-time approach for automatic defect detection from pcbs based on surf features and morphological operations. volume 78, 12 2019. [Accessed on: 19-July-2024] https://www.mdpi.com/2624-7402/4/2/34#:~:text=Whereas%20two%2Dshot%20object%20detectors,refinement%20of%20the%20location%20prediction.

[10] Sunyawit Sassanapitak and Pakorn Kaewtrakulpong. An efficient translation-rotation template matching using pre-computed scores of rotated templates. In *2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, volume 02, pages 1040–1043, 2009. [Accessed on: 19-July-2024] https://ieeexplore.ieee.org/abstract/document/5137223.

[11] Jing Li, Jinan Gu, Zedong Huang, and Jia Wen. Application research of improved yolo v3 algorithm in pcb electronic component detection. volume 9, page 3750, 09 2019. [Accessed on: 26-May-2024], https://ieeexplore.ieee.org/document/9918069.

[12] Haojia Xin, Zibo Chen, and Boyuan Wang. Pcb electronic component defect detection method based on improved yolov4 algorithm. volume 1827, page 012167. IOP Publishing Ltd, 2021. 6th International Conference on Electronic Technology and Information Science (ICETIS 2021), 8-10 January 2021, Harbin, China. [Accessed on: 26-May-2024], https://iopscience.iop.org/article/10.1088/1742-6596/1827/1/012167.

[13] Wang Xuan, Gao Jian-She, Hou Bo-Jie, Wang Zong-Shan, Ding Hong-Wei, and Wang Jie. A lightweight modified yolox network using coordinate attention mechanism for pcb surface defect detection. volume 22, pages 20910–20920, 2022. [Accessed on: 26-May-2024], https://ieeexplore.ieee.org/document/9905499.

[14] Jianfeng Zheng, Xiaopeng Sun, Haixiang Zhou, Chenyang Tian, and Hao Qiang. Printed circuit boards defect detection method based on improved fully convolutional networks. volume PP, pages 1–1, 01 2022. [Accessed on: 26-May-2024], https://www.researchgate.net/publication/364526413_Printed_Circuit_Boards_Defect_Detection_Method_Based_on_Improved_Fully_Convolutional_Networks.

[15] Xing Wu, Yuxi Ge, Qingfeng Zhang, and Dali Zhang. Pcb defect detection using deep learning methods. In *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 873–876, 2021. [Accessed on: 26-May-2024], https://ieeexplore.ieee.org/document/9437846.

[16] Wenkai Huang, Jiafu Wen, Weiming Gan, Lingkai Hu, and Bingjun Luo. Neighborhood correlation enhancement network for pcb defect classification. volume 72, pages 1–11, 2023. [Accessed on: 26-May-2024], https://ieeexplore.ieee.org/document/10041176.

[17] Mengke Li, Naifu Yao, Sha Liu, Shouqing Li, Yongqiang Zhao, and Seong G. Kong. Multi-sensor image fusion for automated detection of defects in printed circuit boards. volume 21,

pages 23390–23399, 2021. [Accessed on: 26-May-2024], https://ieeexplore.ieee.org/document/9517112.

[18] Chinmay U. Parab, Canicius Mwitta, Miller Hayes, Jason M. Schmidt, David Riley, Kadeghe Fue, Suchendra Bhandarkar, and Glen C. Rains. Comparison of single-shot and two-shot deep neural network models for whitefly detection in iot web application. volume 4, pages 507–522, 2022. [Accessed on: 31-May-2024] https://www.mdpi.com/2624-7402/4/2/34#:~:text=Whereas%20two%2Dshot%20object%20detectors,refinement%20of%20the%20location%20prediction.

[19] Kim JY Kang SH. Application of fast non-local means algorithm for noise reduction using separable color channels in light microscopy images. Int J Environ Res Public Health, March 2021. [Accessed on: 31-May-2024]. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8001297/.

[20] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. Prentice Hall, Upper Saddle River, N.J., 2008. [Accessed on: 01-June-2024].

[21] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979. [Accessed on: 01-June-2024]. https://ieeexplore.ieee.org/document/4310076.

[22] X. Binjie and J. Hu. 6 - fabric appearance testing. In Jinlian Hu, editor, *Fabric Testing*, Woodhead Publishing Series in Textiles, pages 148–188. Woodhead Publishing, 2008. [Accessed on: 01-June-2024]. https://www.sciencedirect.com/science/article/pii/B9781845692971500061.

[23] David G. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision*, page 91–110. November 2004. [Accessed on: 01-June-2024]. https://link.springer.com/article/10.1023/B:VISI.0000029664.99615.94.

[24] Guohui Wang, Blaine Rister, and Joseph Cavallaro. Workload analysis and efficient opencl-based implementation of sift algorithm on a smartphone. 12 2013. [Accessed on: 19-July-2024] https://www.researchgate.net/publication/256546531_Workload_Analysis_and_Efficient_OpenCL-based_Implementation_of_SIFT_Algorithm_on_a_Smartphone.

Documents and Websites References

[25] Nazmul Abir. Strategies to address inspector fatigue in pharmaceutical visual inspection. https://www.linkedin.com/pulse/strategies-address-inspector-fatigue-pharmaceutical-visual-abir/, 2021. [Accessed: 11-May-2024].

[26] Robert Keim. What is a printed circuit board (pcb)? https://www.allaboutcircuits.com/technical-articles/what-is-a-printed-circuit-board-pcb/, 2020. [Accessed on: 19-May-2024].

[27] Umar Waseem. Types of printed circuit boards. https://www.wevolver.com/article/types-of-printed-circuit-boards, 2024. [Accessed on: 12-May-2024].

[28] ncabgroup. The history of pcbs. https://www.ncabgroup.com/blog/history-of-pcbs/, 2024. [Accessed on: 19-May-2024].

[29] Alam Mamtaz. The history of printed circuit board pcb 1880 – present. https://how2electronics.com/history-printed-circuit-board-pcb-nextpcb/, 2023. [Accessed on: 19-May-2024].

[30] mclpcb. History of pcb innovations and their impact. https://www.mclpcb.com/blog/history-of-pcbs/. [Accessed on: 19-May-2024].

[31] printedcircuits. The history of pcbs. https://www.printedcircuits.com/blog/history-of-pcbs/. [Accessed on: 19-May-2024].

[32] elecrow. All about multilayer pcbs you should know. https://www.elecrow.com/all-about-multilayer-pcbs-you-should-know-printed-circuit-board. [Accessed on: 19-May-2024].

[33] orient display. Doppelseitige leiterplattenherstellung. https://www.orientdisplay.com/de/knowledge-base/pcb-basics/double-sided-pcb-manufacturing/, 2022. [Accessed on: 19-May-2024].

[34] alphaemscorp. Pcb circuit board test and inspection. https://alphaemscorp.com/pcb-circuit-board-test-inspection/pcb-circuit-board-test-and-inspection/. [Accessed on: 24-May-2024].

[35] Matric Group. 7 pcb testing methods you need to know. https://blog.matric.com/pcb-testing-methods, 2023. [Accessed on: 24-May-2024].

[36] Swimbi Electronic Manufacturing. How does an automated optical inspection (aoi) work? https://www.linkedin.com/pulse/how-does-automated-optical-inspection-qmzxc/, 2023. [Accessed on: 24-May-2024].

[37] Hillman Curtis. How does the pcb x ray machine work? https://hillmancurtis.com/pcb-x-ray-inspection/. [Accessed on: 24-May-2024].

[38] Jack Olson. Circuit board design for in-circuit testers. https://resources.altium.com/p/circuit-board-design-circuit-testability, 2020. [Accessed on: 24-May-2024].

[39] Muhammad Sufyan. A comprehensive introduction to flying probe test, its essential components, advantages, limitations, and application cases. https://www.wevolver.com/article/flying-probe-test-an-extensive-guide-to-the-technology-and-applications, September 2023. [Accessed on: 24-May-2024].

[40] GTS TEST SOLUTIONS. Functional testing (fct) of printed circuit boards (pcbs) or electronic devices. https://gts-online.net/en/applications/functional-testing-fct/. [Accessed on: 24-May-2024].

[41] OpenCV. Introduction to surf (speeded-up robust features). https://docs.opencv.org/3.4/df/dd2/tutorial_py_surf_intro.html. [Accessed on: 19-July-2024].

[42] OpenCV. Orb (oriented fast and rotated brief). https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html. [Accessed on: 19-July-2024].

[43] OpenCV. Brief (binary robust independent elementary features). https://docs.opencv.org/3.4/dc/d7d/tutorial_py_brief.html. [Accessed on: 19-July-2024].

[44] OpenCV. Fast algorithm for corner detection. https://docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html. [Accessed on: 19-July-2024].

[45] OpenCV. Template matching. https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html. [Accessed on: 01-June-2024].

[46] OpenCV. Template matching. https://docs.opencv.org/3.4/de/da9/tutorial_template_matching.html. [Accessed on: 01-June-2024].

[47] Python Software Foundation. Python. https://www.python.org/, 2024. [Accessed on: 27-May-2024].

[48] JetBrains. PyCharm: The Python IDE for Professional Developers. https://www.jetbrains.com/pycharm/, 2024. [Accessed on: 27-May-2024].

[49] Google. Google Colaboratory. https://colab.research.google.com/, 2024. [Accessed on: 31-May-2024].

[50] Lev Craig. convolutional neural network (cnn)w. https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network. [Accessed on: 31-May-2024].

[51] ultralytics. Yolov8: architecture. https://yolov8.org/yolov8-architecture/. [Accessed on: 31-May-2024].

[52] VK. Yolov8 architecture & cow counter with region based dragging using yolov8. https://medium.com/@VK_Venkatkumar/yolov8-architecture-cow-counter-with-region-based-dragging-using-yolov8-e75b3ac71ed November 2023. [Accessed on: 31-May-2024].

[53] Akshit Mehra. Understanding yolov8 architecture, applications & features. https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/, April 2023. [Accessed on: 31-May-2024].

[54] Ashley Walker Robert Fisher, Simon Perkins and Erik Wolfart. Gaussian smoothing. https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm. [Accessed on: 31-May-2024].

[55] geeksforgeeks. Python — bilateral filtering. https://www.geeksforgeeks.org/python-bilateral-filtering/, January 2023. [Accessed on: 31-May-2024].

[56] Justin Liang. Canny edge detection. https://justin-liang.com/tutorials/canny/. [Accessed on: 31-May-2024].

[57] OPEN CV. Canny edge detection. https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html. [Accessed on: 31-May-2024].

[58] Krishna Rohit. Coding canny edge detection algorithm from scratch in python. https://medium.com/@rohit-krishna/coding-canny-edge-detection-algorithm-from-scratch-in-python-232e1fdceac7, January 2023. [Accessed on: 31-May-2024].

[59] Sahir Sofiane. Canny edge detection step by step in python — computer vision. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123, January 2019. [Accessed on: 31-May-2024].

[60] OpenCV. Image thresholding. https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html. [Accessed on: 01-June-2024].

[61] Jean Serra. Courses on mathematical morphology. https://people.cmm.minesparis.psl.eu/users/serra/cours/index.htm. [Accessed on: 01-June-2024].

[62] OpenCV. Feature matching. https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html#:~:text=FLANN%20stands%20for%20Fast%20Library,than%20BFMatcher%20for%20large%20datasets. [Accessed on: 20-July-2024].

[63] OpenCV. Feature matching with flann. https://docs.opencv.org/4.x/d5/d6f/tutorial_feature_flann_matcher.html. [Accessed on: 20-July-2024].

[64] GISGeography. How gps receivers work – trilateration vs triangulation. https://gisgeography.com/trilateration-triangulation-gps/, March 2024. [Accessed on: 01-June-2024].

[65] Python Software Foundation. pickle — python object serialization. [Accessed: 2024-06-28] https://docs.python.org/3/library/pickle.html.

[66] Alex Clark. Pillow (PIL Fork) Documentation. https://pillow.readthedocs.io/en/stable/, 2015. [Accessed on: 27-May-2024].

[67] Riverbank Computing. PyQt6 Reference Guide. https://www.riverbankcomputing.com/static/Docs/PyQt6/, 2021. [Accessed on: 27-May-2024].

[68] J. D. Hunter. Matplotlib: A 2d graphics environment. https://www.computer.org/csdl/magazine/cs/2007/03/c3090/13rRUwbJD0A, 2007. [Accessed on: 27-May-2024].

[69] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python, 2014. [Accessed on: 31-May-2024].

[70] Asmaa Mirkhan. Yolo algorithm: Real-time object detection from a to z. https://kili-technology.com/data-labeling/machine-learning/yolo-algorithm-real-time-object-detection-from-a-to-z#what-is-yolo? [Accessed on: 31-May-2024].

[71] Bharath K. Object detection algorithms and libraries. https://neptune.ai/blog/object-detection-algorithms-and-libraries, April 2024. [Accessed on: 31-May-2024].

[72] keylabs. Under the hood: Yolov8 architecture explained. https://keylabs.ai/blog/under-the-hood-yolov8-architecture-explained/, December 2023. [Accessed on: 31-May-2024].

[73] ultralytics. Ultralytics yolov8 tasks. https://docs.ultralytics.com/tasks/. [Accessed on: 31-May-2024].

[74] ultralytics. Oriented bounding box (obb) datasets overview. https://docs.ultralytics.com/datasets/obb/, November 2023. [Accessed on: 31-May-2024].

[75] José Murta. Raspberry pi. https://www.flickr.com/photos/jgustavoam/albums/72157632650634969/. [Accessed on: 08-July-2024].

Appendix

Table A.1: Inspection time

Board	Coloured Image	Grey Image	Filtered Coloured Image	Filtered Grey Image
A - Traditional	0.77	0.75	15.55	4.37
A - YOLO	3.05	2.87	32.99	11.56
A - Combined	5.04	4.93	16.35	8.1
B - Traditional	1.12	0.61	14.24	4.37
B - YOLO	4.91	4.74	34.33	12.43
B - Combined	7.26	3.45	15.99	8.17
C - Traditional	0.87	0.63	18.44	5.57
C - YOLO	3.64	3.2	61.81	22.14
C - Combined	5.0	4.81	19.87	9.23
D - Traditional	0.88	0.64	17.57	5.55
D - YOLO	3.79	3.33	62.5	23.42
D - Combined	5.14	4.93	19.92	9.31
E - Traditional	0.87	0.62	22.31	5.86
E - YOLO	3.7	3.39	61.04	22.59
E - Combined	5.11	4.86	24.59	9.74
F - Traditional	0.92	0.65	22.21	6.79
F - YOLO	6.33	3.18	65.5	23.91
F - Combined	5.05	4.79	24.59	10.82
G - Traditional	1.02	0.70	18.33	5.94
G - YOLO	3.59	3.24	67.14	24.54
G - Combined	5.09	4.87	20.37	9.52
H - Traditional	1.06	0.70	22.78	5.73
H - YOLO	3.56	3.2	68.56	24.42
H - Combined	5.02	4.69	25.16	9.26
I - Traditional	1.06	0.74	18.10	6.71
I - YOLO	3.84	3.28	64.76	22.83
I - Combined	5.15	4.98	20.27	10.8
J - Traditional	1.08	0.73	19.97	6.97
J - YOLO	3.8	3.37	65.91	23.29
J - Combined	5.14	5.08	22.35	11.04
K - Traditional	0.99	0.71	24.45	8.51
K - YOLO	3.38	3.11	65.24	23.83
K - Combined	5.15	4.83	27.14	12.54
L - Traditional	0.62	0.54	14.27	4.33
L - YOLO	2.92	2.91	24.08	8.73
L - Combined	4.97	4.89	16.03	8.07
M - Traditional	1.02	0.61	15.49	4.86
M - YOLO	3.12	2.85	32.23	11.27
M - Combined	4.89	4.72	15.96	7.9
N - Traditional	0.79	0.6	14.46	4.35
N - YOLO	3.06	2.8	32.26	11.18
N - Combined	4.93	4.70	15.79	8.18

Table A.2: Inspection failure

Board	**Coloured Image**	**Grey Image**	**Filtered Coloured Image**	**Filtered Grey Image**
A - Traditional	0	0	0	1
A - YOLO	0	0	0	0
A - Combined	1	1	1	1
B - Traditional	0	0	2+(1 False True)	2+(1 False True)
B - YOLO	0	0	0	0
B - Combined	1	1	1	1
C - Traditional	1+(1 False True)	1+(1 False True)	0+(1 False True)	0+(1 False True)
C - YOLO	3	3	3	3
C - Combined	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)
D - Traditional	1+(1 False True)	1+(1 False True)	0+(1 False True)	0+(1 False True)
D - YOLO	4	4	3	3
D - Combined	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)
E - Traditional	0+(1 False True)	0+(1 False True)	0+(1 False True)	1+(1 False True)
E - YOLO	1	1	1	1
E - Combined	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)
F - Traditional	0+(1 False True)	0+(1 False True)	0+(1 False True)	0+(1 False True)
F - YOLO	1	1	1	1
F - Combined	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)
G - Traditional	0+(1 False True)	2+(1 False True)	0+(1 False True)	2+(1 False True)
G - YOLO	2	2	2	2
G - Combined	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)
H - Traditional	0+(1 False True)	2+(1 False True)	1+(1 False True)	2+(1 False True)
H - YOLO	1	1	1	1
H - Combined	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)
I - Traditional	0+(1 False True)	0+(1 False True)	0+(1 False True)	2+(1 False True)
I - YOLO	3	3	3	3
I - Combined	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)
J - Traditional	2+(1 False True)	0+(1 False True)	1+(1 False True)	2+(1 False True)
J - YOLO	2	2	2	2
J - Combined	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)	1+(1 labelling error)
K - Traditional	1	1	0	1
K - YOLO	3	3	3	3
K - Combined	1	1	1	1
L - Traditional	1	0	0	1
L - YOLO	0	0	0	0
L - Combined	1	1	1	1
M - Traditional	0	1	2+(1 False True)	2
M - YOLO	0	0	0	0
M - Combined	1	1	1	1
N - Traditional	0	0	0	0
N - YOLO	0	0	0	0
N - Combined	1	1	1	1

Table A.3: Frequency of accessing secondary templates in inspection process

Board	**Coloured Image**	**Grey Image**	**Filtered Coloured Image**	**Filtered Grey Image**
A - Traditional	2	1	1	0
B - Traditional	5	2	1	1
C - Traditional	7	6	5	5
D - Traditional	9	9	5	6
E - Traditional	9	7	6	7
F - Traditional	14	12	6	9
G - Traditional	13	13	7	10
H - Traditional	15	13	8	9
I - Traditional	17	15	7	13
J - Traditional	17	17	11	14
K - Traditional	19	17	11	15
L - Traditional	0	0	0	0
M - Traditional	2	1	2	0
N - Traditional	2	2	1	1