Department of Electronic & Telecommunication Engineering
University of Moratuwa, Sri Lanka.

Design Documentation

Computer Vision-Based PCB Defects Detection and Component Testing Device

L.J.J.P Silva
210608J

Submitted in partial fulfillment of the requirements for the module
EN 2160 Electronic Design Realization

30th March 2024

Contents

1 Introduction

The project **"Computer Vision-Based PCB Defects Detection and Component Testing"** is positioned to set a new standard in PCB manufacturing quality control. Through leveraging advanced image segmentation and deep learning models, the project seeks to substantially improve the detection of common PCB defects such as spurs, mouse bites, scratches, missing holes, open circuits, short circuits, and inaccuracies in component placement and orientation. The goal is to offer a cost-effective solution that maintains high accuracy and reliability, making advanced PCB inspection accessible to small and medium-sized industries.

The document outlines the research and development process undertaken to implement a robust device, encompassing background research, exploration of techniques, selection of frameworks and tools, and the initial setup for training and testing models.

2 System Requirements

Before proceeding with the setup process, ensure your system meets the following requirements:

- **Operating System**:

  The algorithm is adaptable and can be implemented on any platform that supports deep learning frameworks, such as in Google Colab. Therefore to train the model Google Colab is preferable.

  Creating the web app is not based on any specific operating system, such as Windows, Linux or macOS.

- **Hardware Requirements:**:

  - **CPU:** Google Colab provides a multi-core CPU suitable for handling preprocessing tasks and managing the overall execution of the algorithm.
  - **GPU:** Google Colab offers GPU support, which can significantly accelerate training and inference processes. Ensure that you select a GPU runtime in Colab to take advantage of this to train the model.

- **Memory (RAM):**

  Google Colab provides ample system memory (RAM), which is essential when working with large datasets and training deep neural networks. As our dataset and model size are within the limits of the available RAM in Colab, Colab is an ideal platform to use.

- **Storage Space:**

  Use Google Drive to store the dataset, trained models, and any intermediate files generated during the training and evaluation processes. Make sure to mount your Google Drive in Colab to access the necessary storage.

3 Training the Model to Detect Defects in PCBs

3.1 Background Research 1/3/2024

Firstly, an exploration was undertaken to ascertain the availability of pre-trained models designed for the detection of Printed Circuit Board (PCB) defects. Unfortunately, none were identified that met the specific criteria requisite for the task.

Consequently, the decision was made to train a model. This necessitated a comprehensive search for diverse datasets suitable for facilitating the training process.

Several datasets were identified during this search:

1. A dataset sourced from Kaggle, comprising 1386 images depicting six distinct types of PCB defects.

   The dataset can be accessed via the following link:

   Kaggle PCB Defects Dataset.

2. A dataset obtained from Label Box, denoted as DeepPCB, containing 226 images showcasing various PCB defects.

   The dataset can be accessed via the following link:

   DeepPCB: Printed circuit board (PCB) defect dataset.

3. A dataset acquired from Roboflow, titled "PCB by Hubei University Of Technology," encompassing 9669 images. Notably, this dataset was deemed considerably well-balanced in terms of representation across different defect categories.

   The dataset can be accessed via the following link:

   Roboflow: Printed circuit board (PCB) defect dataset.

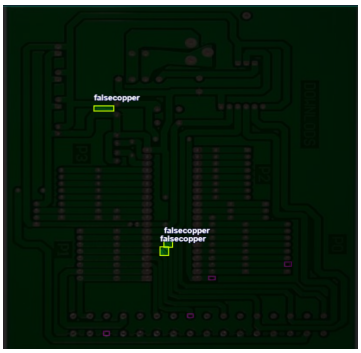
   The images are slit as:

   - **Training Set:** 8916 (97%)
   - **Valid Set:** 753 (8%)

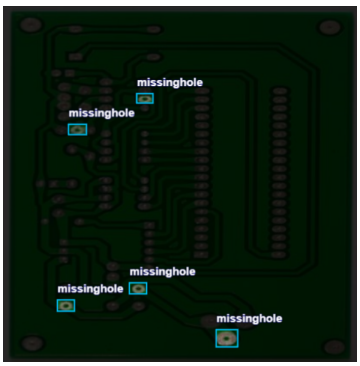
   It comprised of 8 types of defects:

   (a) falsecopper
   (b) missinghole
   (c) mousebite
   (d) opencircuit
   (e) pinhole
   (f) scratch
   (g) shortcircuit
   (h) spur

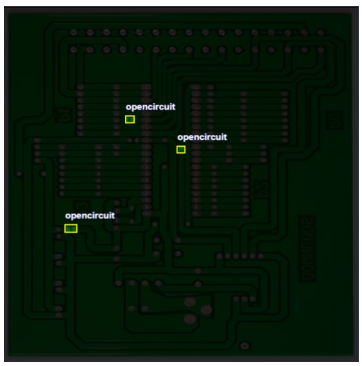
Following careful consideration, the dataset sourced from Roboflow was selected as the preferred choice for model training. This decision was primarily motivated by the dataset's expansive size, which promises enhanced accuracy due to the ample representation of various defect scenarios. Additionally, the dataset's compatibility with the YOLOv8 architecture was perceived as advantageous, potentially facilitating a smoother and more efficient training process.



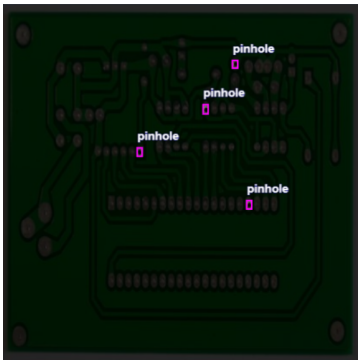
(a) Falsecopper



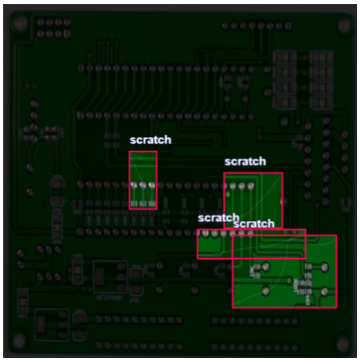
(b) Missing Hole



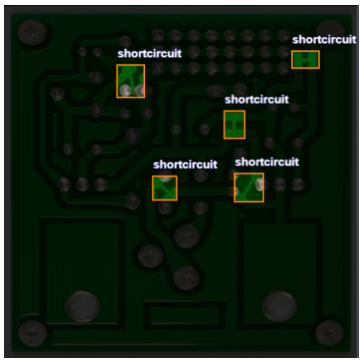
(c) Open Circuit



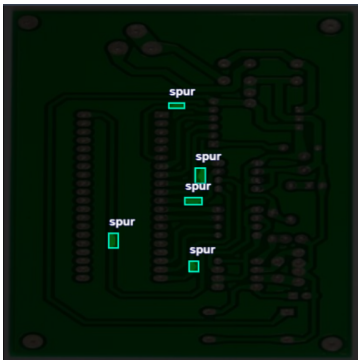
(d) Pinhole



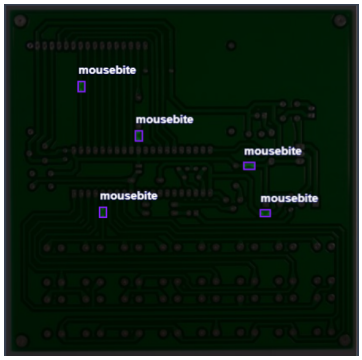
(e) Scratch



(f) Short Circuit



(g) Spur



(h) Mousebite

Figure 1: The 8 types of Defects Detected

3.2 Initial Setup 6/3/2024

Setting up the environment was the next thing. It was chosen to test the models we were given and the models we discovered as we continued our investigation with the following configuration.

- **Google Colab:**

  For a number of strong reasons, we decided to use Google Colab to implement the models for testing and training:

  - Google Colab offers a free, publicly available cloud-based environment with strong GPUs that makes model training faster than on local computers.
  - Colab provides a smooth interface with well-known deep learning frameworks like PyTorch and TensorFlow, making setup easier and enabling effective implementation of complicated models.
  - Easy data access and storage are made possible by its interface with Google Drive, which streamlines the entire process from data preparation to model evaluation.
  - In general, using Google Colab to develop the models guarantees effective and scalable model development for image segmentation tasks.

- **Yolov8:**

  - **Efficiency, Accuracy, and Speed:**
    YOLOv8 is specifically selected due to its efficiency, accuracy, and speed. When dealing with real-world applications like PCB defect detection, these factors are crucial.
  - **Customization with Your Own Data:**
    YOLOv8 allows you to train custom models using your own data. You can fine-tune the model to specifically detect PCB defects by providing annotated images of defects.
  - **Robust Object Detection:**
    YOLOv8 excels in detecting multiple objects within an image simultaneously. It divides the image into a grid and predicts bounding boxes and class probabilities for each grid cell. This grid-based approach ensures robust detection even when objects overlap or are close together.
  - **Community Support and Resources:**
    YOLOv8 has gained popularity, resulting in a strong community of developers, researchers, and practitioners. This community support provides valuable resources, tutorials, and troubleshooting assistance.

3.3 Training the Model 15/3/2024

The following steps were followed in order to train the model from Yolov8 using the Dataset obtained by Roboflow.

1. Enable GPU:

   Google Colab offers GPU support, which can significantly accelerate training and inference processes, especially for larger datasets. Ensure that you select a GPU runtime in Colab to take advantage of this.

   (a) **Open Your Notebook**: Open your Google Colab notebook where you plan to run your code.

   (b) **Access Runtime Settings**:

      i. Click on the `Runtime` menu at the top of the screen.

      ii. Select `Change runtime type` from the dropdown menu.

   (c) **Select GPU**:

      i. In the `Runtime type` dropdown, select `Python 3`.

      ii. In the `Hardware accelerator` dropdown, select `GPU`.

      iii. Click `Save` to apply the changes.

2. Verify GPU Activation:

   After enabling the GPU, you can verify its availability by running the following command in a code cell:

```
#jkjsd
!nvidia-smi
```

   This command will display information about the GPU, including its model, memory usage, and other details.

3. Install ultralytics package:

   By installing the ultralytics package, you gain access to tools and functions to create, train, and evaluate YOLOv8 models for tasks such as object detection, which includes detecting PCB defects.

   To install the package run the following command.

```
!pip install ultralytics
```

4. Importing other packages

   Run the code below to complete the setup.

```
from ultralytics import YOLO
import os
from IPython.display import display, Image
from IPython import display
display.clear_output()
!yolo mode=checks
```

- It imports the YOLO class from the ultralytics package, allowing you to create and use YOLOv8 models.
- The os module is imported for interacting with the operating system, such as file and directory operations.
- It imports the display and Image functions from IPython to show images and other outputs in Jupyter notebooks.
- It lears the current output in a Jupyter notebook to make the cell output cleaner.
- Finally, the command yolo mode=checks is executed to run a YOLO system check, which verifies that the environment is set up correctly for using YOLOv8.

5. Train YOLOv8 Model on the Dataset

   To obtain the download code from Roboflow, follow these steps:

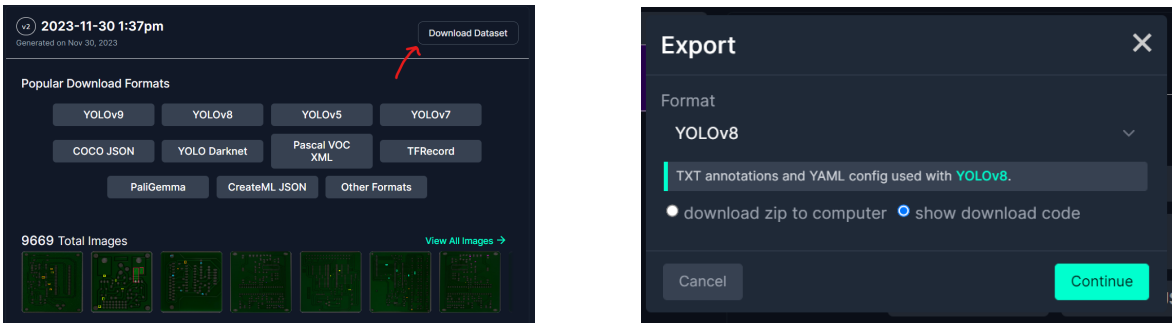
   (a) **Sign in or Sign up**: Go to the Roboflow website and sign in to your account. If you don't have an account, you'll need to sign up for one.

   (b) **Find the Dataset**: Click on the link to access find dataset for PCB defects. PCB defects dataset

   (c) **Generate the Code**: Once you're in the project, look for the export options or download options. Roboflow provides various options for exporting datasets, including download code snippets for popular deep learning frameworks like YOLOv5, TensorFlow, PyTorch, etc.

   (d) **Select YOLOv8**: Choose the YOLOv8 option or any other compatible option that suits your needs.

   (e) **Copy the Code**: After selecting YOLOv8, you'll be provided with a code snippet similar to the one you posted. Copy the code snippet.



(a) Click on Download (b) Choose the format as Yolov8

Figure 2: Copying the download code for the dataset

   (f) **Use the Code**: Paste the copied code into your Coloab Project.

```
!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="YOUR-API-KEY")
project = rf.workspace("nicolai-hoirup-nielsen").project("cup-
↪ detection-v2")
dataset = project.version(3).download("yolov8")
```

6. Start training:

   To start training run the code below.

```
!yolo task=detect mode=train model=yolov8m.pt data=dataset.
↪ location}/data.yaml epochs=20 imgsz=640
```

   - **task=detect**: Specifies the task to be performed, which in this case is detection (identifying objects within images).
   - **mode=train**: Indicates that the YOLOv8 model will be trained.
   - **model=yolov8m.pt**: Specifies the model architecture to be used for training. In this case, it's the YOLOv8m architecture, represented by the model file `yolov8m.pt`.
   - **data={dataset.location}/data.yaml**: Specifies the location of the data configuration file (`data.yaml`) used for training the model. {dataset.location} is likely a placeholder for the location of your dataset.
   - **epochs=20**: Sets the number of training epochs to 20. An epoch is one complete pass through the entire dataset during training.
   - **imgsz=640**: Sets the input image size to 640x640 pixels. Larger input sizes can improve detection accuracy but require more computational resources.

   The model is now trained.

7. Download the trained model

3.3.1 Overview of the Results

The overall results of the trained model demonstrate promising outcomes, indicating its suitability for effectively detecting defects on the PCB.

- **train/box_loss**:

  This graph represents the bounding box loss during training. The x-axis likely corresponds to the number of training iterations, while the y-axis represents the loss value. The downward trend indicates that the model is improving in terms of accurately predicting bounding boxes around defects. As the iterations progress, the loss decreases, which is a positive sign.

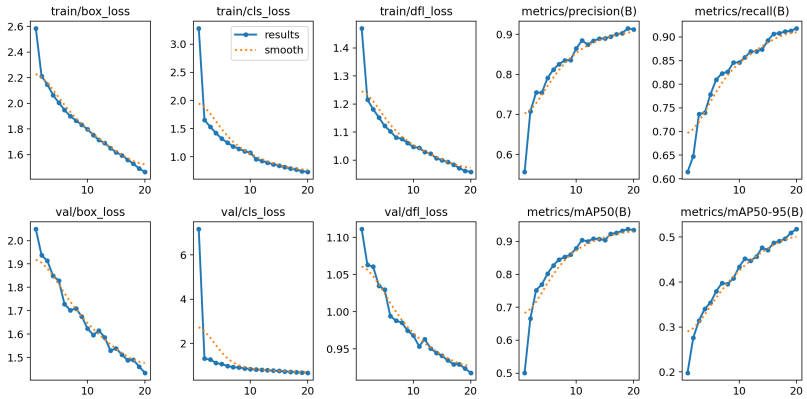


Figure 3: Output Results

- **train/cls_loss**:

  The classification loss can be expressed as:

$$\mathcal{L}_{\text{cls}} = -\frac{1}{N_{\text{defect}}} \sum_{i=1}^{N_{\text{defect}}} \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right) \tag{1}$$

  This graph shows the classification loss during training. It measures how well the model classifies defects. Similar to the previous graph, a decreasing trend is desirable. It means the model is learning to classify defects more accurately.

- **train/obj_loss**: The object loss graph reflects how well the model detects objects (defects) within the image. Again, a decreasing trend indicates that the model is improving its object detection capabilities.

- **val/box_loss**: The validation box loss graph represents the same concept as the first graph but on the validation dataset. The sharp decrease followed by a plateau suggests that the model initially improved significantly during validation but then stabilized.

- **val/cls_loss**: The validation classification loss graph mirrors the second graph but on the validation data. A decreasing trend here indicates that the model's classification accuracy is improving during validation.

- **val/diff_iou**: This graph shows the intersection over union (IoU) metric during validation. IoU measures the overlap between predicted bounding boxes and ground truth boxes. An increasing trend means better alignment between predicted and actual defects.

The Confusion Matrix also presents a positive outcome, showcasing the accurate identification of defects.

Below are additional results presented to provide a comprehensive insight into the accuracy of the trained model.

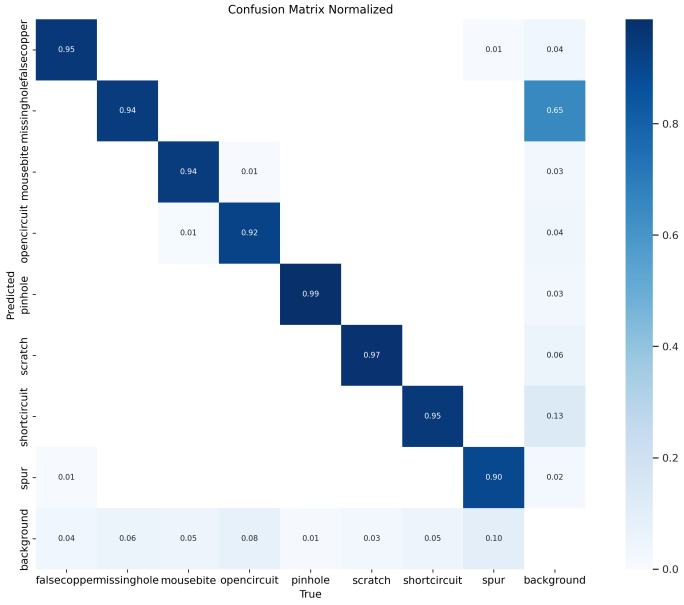
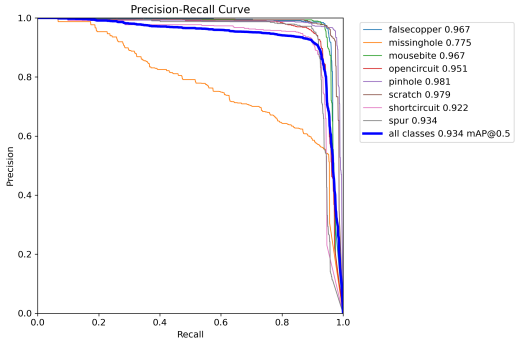
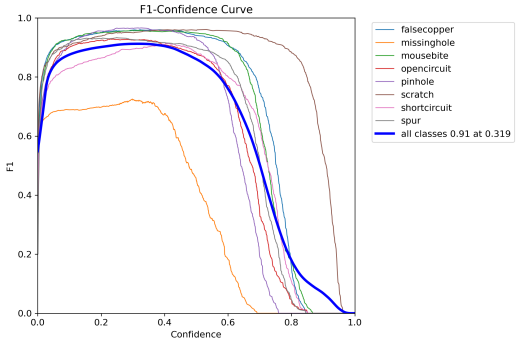


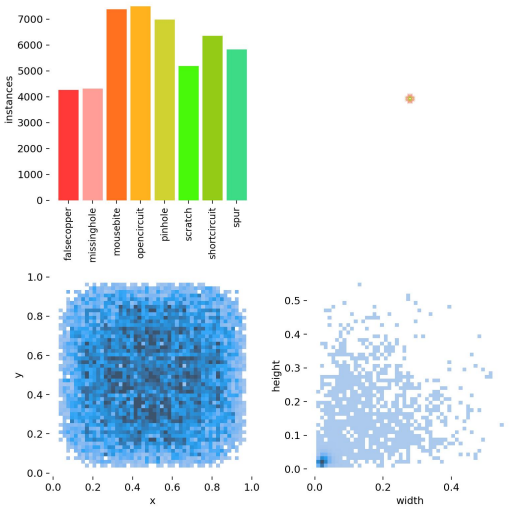
Figure 4: Confusion Matrix



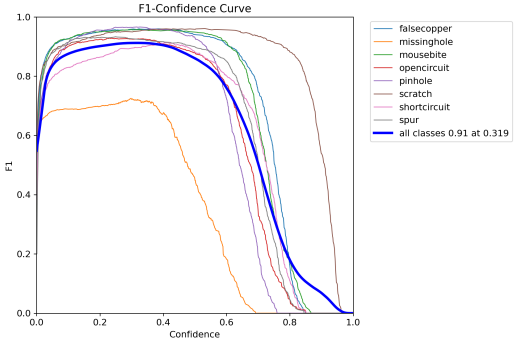
(a) Precision Recall Curve



(b) Confidence Curve



(c) Spreading of Data



(d) F1 Curve

Figure 5: Model Results

3.3.2 Validating the Model

This command is executing a YOLO task in detection mode on a validation dataset using a specific model file. It's part of the evaluation process to assess the performance of the YOLO model on detecting objects specified in the data.yaml file.

Run this on your Colab Project.

```
!yolo task=detect mode=predict model=/content/runs/detect/train/
↪ weights/best.pt conf=0.5 source={dataset.location}/test/images
↪ save_txt=true save_conf=true
```

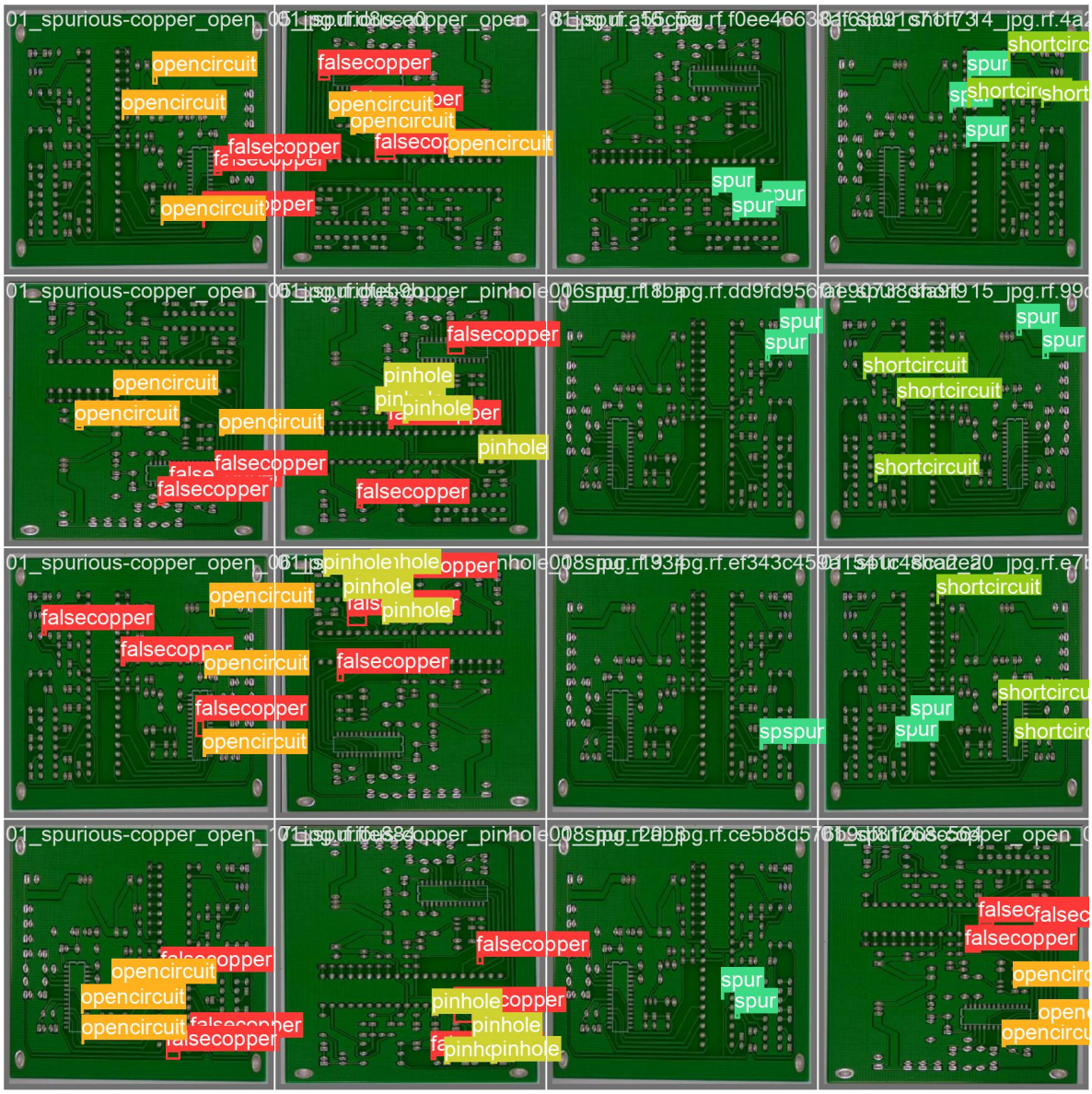
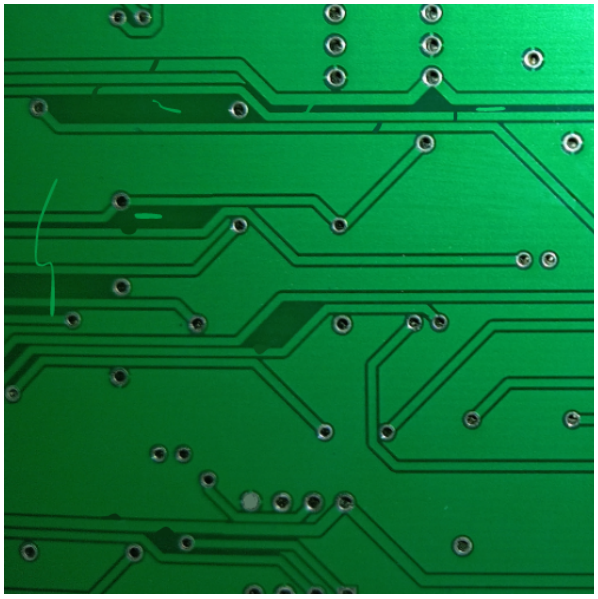


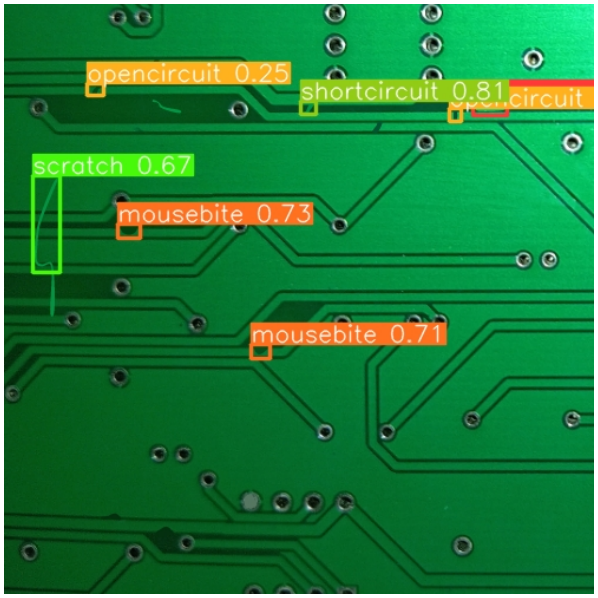
Figure 6: Validation Results

3.3.3 Testing on an Image

The trained model was tested on an actual image of a PCB captured by a camera and some defects were marked on the image. The model gave successful results, identifying most of the defects in image.



(a) PCB with defects



(b) PCB with identified defects

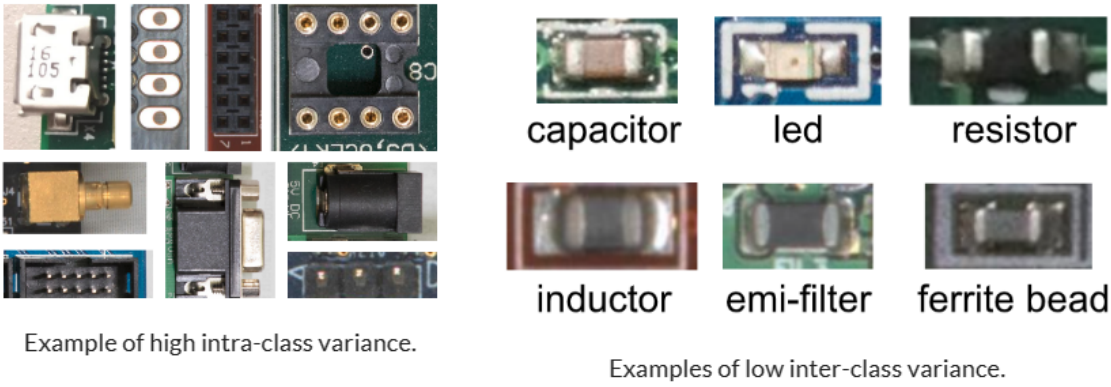
Figure 7: Identification of the defects on a real PCB

4 Identifying Components' Placement and Orientation Detection

4.1 Background Research 25/3/2024

Various approaches were investigated to identify defects in the placement and alignment of components on the PCB.

Initially, an image classification model was trained to detect individual components on the PCB. However, this method encountered challenges due to the substantial variations both between different types of components (inter-class variance) and within the same type of component (intra-class variance). These variations made it difficult for the classification model to accurately differentiate between different components and classify them correctly.



Example of high intra-class variance.

Examples of low inter-class variance.

Figure 8: Inter Class and Intra Class Varience

As a result, it was determined that this approach was not the most effective for detecting defects in the placement and orientation of components on the PCB.

Subsequently, the examination of defects led to consideration of anomaly detection as a potential solution. This approach promised to provide detailed insights into both the placement and orientation of components on the PCB.

4.2 Initial Setup 2/4/2024

OpenCV was recognised as the ideal framework to use. Here are some potential reasons why OpenCV is chosen over other frameworks this task:

- OpenCV

  OpenCV was recognised as the ideal framework to use. Here are some potential reasons why OpenCV is chosen over other frameworks this task:

  - **Comprehensive Functionality:** OpenCV offers a wide range of built-in functions and algorithms for various computer vision tasks, such as image/video processing, object detection, feature extraction, and more.

- **Community Support:** OpenCV has a large and active community of users and developers, which means there are plenty of resources, tutorials, and forums available for help and learning.
- **Cross-platform Compatibility:** OpenCV is cross-platform and can run on various operating systems such as Windows, Linux, macOS, Android, and iOS, making it versatile for different development environments.
- **Integration with Other Libraries:** OpenCV can be easily integrated with other popular libraries and frameworks like NumPy, SciPy, and TensorFlow, allowing for seamless integration into existing workflows or pipelines.

Overall, the choice of using OpenCV depends on the specific requirements of the project, the familiarity of the developer with the library, and the performance and functionality needed for the task at hand.

- VS Code:

  Visual Studio Code (VS Code) is a free and open-source code editor developed by Microsoft. Becaouse of vast features such as it's cross-platform and widely used by developers for writing, editing, and debugging code. VS Code features IntelliSense for code completion, built-in Git integration, a rich ecosystem of extensions, debugging support, an integrated terminal, and extensive customization options.

  In this section, we present a method for comparing two images and detecting defects in the placement and orientation of components on a Printed Circuit Board (PCB).

4.3 Methodology

The method is presented here for comparing two images and detecting defects in the placement and orientation of components on a Printed Circuit Board (PCB).

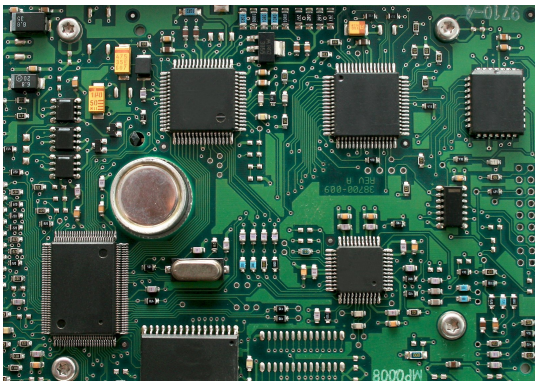
- **Image Conversion:**
  - The reference image and source image are initially converted from the RGB color space to the BGR color space using OpenCV's `cv2.cvtColor` function. This conversion ensures consistency in image processing.
- **Grayscale Conversion:**
  - After the color space conversion, both images are further converted to grayscale using `cv2.cvtColor` with the `cv2.COLOR_BGR2GRAY` parameter. Grayscale images simplify the comparison process by focusing only on intensity differences.
- **Difference Calculation:**
  - The absolute difference between the grayscale representations of the two images is computed using OpenCV's `cv2.absdiff` function. This operation highlights regions where the images significantly differ in intensity.
- **Thresholding:**

    - To further emphasize the differences, a thresholding technique is applied to the difference image. We use OpenCV's `cv2.threshold` function to convert the difference image into a binary image, where pixel values above a certain threshold are set to white (255) and below the threshold are set to black (0).

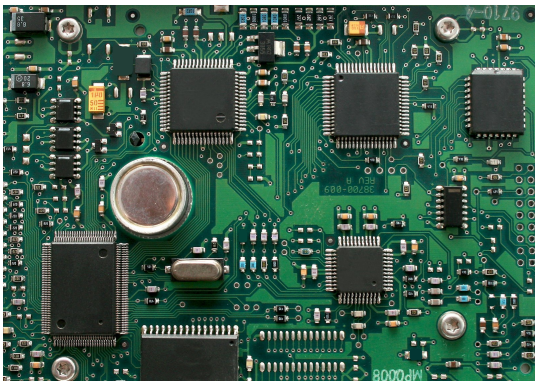
- **Contour Detection:**
    - The contours of the differing regions in the thresholded image are detected using OpenCV's `cv2.findContours` function. Contours represent the boundaries of these regions and are used to identify the location and extent of defects on the PCB.
- **Defect Visualization:**
    - Finally, rectangles are drawn around the detected contours using OpenCV's `cv2.rectangle` function. These rectangles highlight the areas where defects are present on the source image, providing a visual representation of the detected anomalies.

4.4 Results

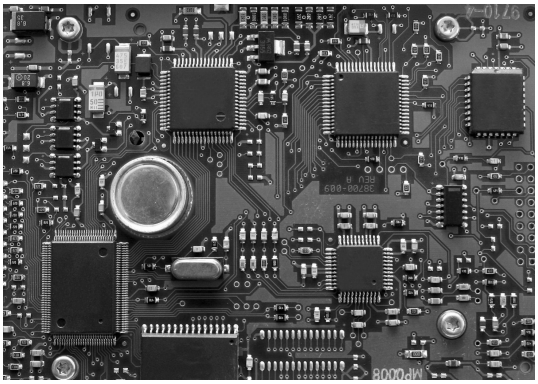
The source image with highlighted defect regions is displayed below:



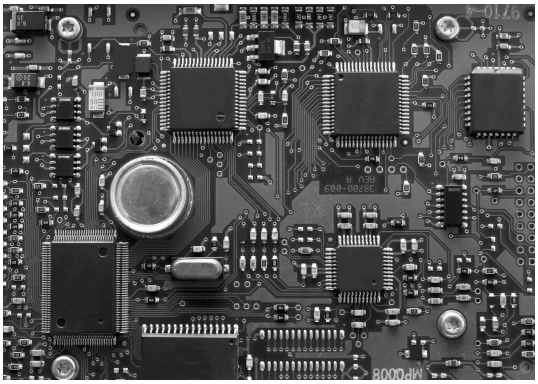
(a) Reference PCB



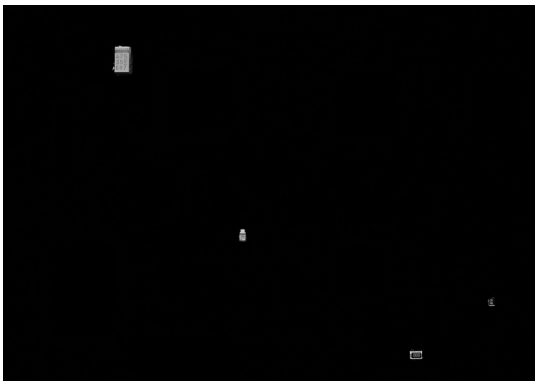
(b) PCB with defects



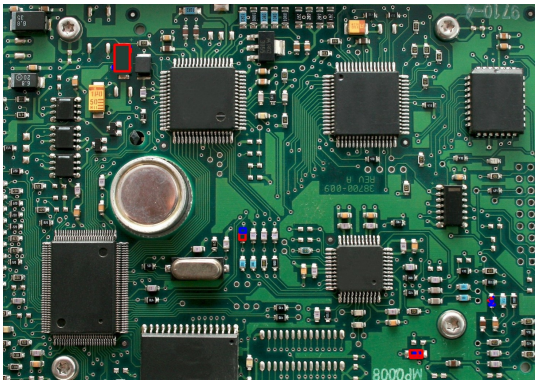
(c) Gray scale of the Reference PCB



(d) Gray scale of the PCB with defects



(e) Gray scale Difference



(f) Defected PCB with Defects Highlighted

Figure 9: Components Defects Detection Result

The proposed method effectively compares two images and identifies defects in the placement and orientation of components on a PCB. By highlighting differing regions between a reference image and a source image, our approach facilitates the rapid detection and visualization of PCB defects, aiding in quality control and manufacturing processes.

Here is the code to detect the anomalies.

```
 reference_img = cv2.cvtColor(np.array(reference_img), cv2.
↪ COLOR_RGB2BGR)
 source_img = cv2.cvtColor(np.array(source_img), cv2.
↪ COLOR_RGB2BGR)

 # Convert images to grayscale
 gray1 = cv2.cvtColor(reference_img, cv2.COLOR_BGR2GRAY)
 gray2 = cv2.cvtColor(source_img, cv2.COLOR_BGR2GRAY)

 # Compute absolute difference between the two images
 difference = cv2.absdiff(gray2, gray1)

 # Apply thresholding to highlight the differences
             _, thresholded = cv2.threshold(difference, 10, 255,
↪  cv2.THRESH_BINARY)
             # Find contours of the differences
             contours, _ = cv2.findContours(thresholded, cv2.
↪ RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
             # Draw rectangles around the differing regions
             for contour in contours:
                 x, y, w, h = cv2.boundingRect(contour)
                 source_img = cv2.cvtColor(np.array(source_img),
↪  cv2.COLOR_BGR2RGB)
                 cv2.rectangle(source_img, (x, y), (x + w, y + h
↪ ), (255, 0, 0), 5)


             st.image(source_img, caption='Tested PCB')
```

5 Schematic Design - Version A 10/4/2024

5.1 Component Selection

For the PCB inspection device, the following modules have been identified as the necessities.

1. Microcontroller
2. Camera
3. Lighting System
4. Power Supply

5.1.1 Microcontroller: ESP-WROOM-32

ESP-WROOM-32 Datasheet

The ESP32S microcontroller has been selected primarily due to its integrated WiFi capabilities, which are essential for transmitting images to a computer at a high data rate. In addition to WiFi, the ESP32S also includes Bluetooth functionality, offering versatile connectivity options.

- **WiFi and Bluetooth Integration:**
  - Integrated 2.4 GHz WiFi for high-speed image transfer.

    Data rate = 50 KB/frame × 15 frames/second = 750 KB/second ≈ 6 Mbps

    The WiFi is able to handle this data rate.
- **Processing Power:**
  - Dual-core Tensilica LX6 processor.
  - Operates at a clock speed of up to 240 MHz.

    Clock cycles per second = 15 FPS×1,000,000 cycles/frame = 15,000,000 cycles/second

    Add some overhead for buffering, network stack handling, and other tasks. Typically, this might double the required clock rate:

    $$15 \text{ MHz} \times 2 = 30 \text{ MHz}$$

    Sufficient processing power for image data handling and efficient WiFi transmission.
- **Memory:**
  - 520 KB of SRAM to support memory-intensive operations.

This combination of integrated WiFi, Bluetooth, robust processing power, and expanded storage solutions makes the ESP32S an ideal choice for the PCB inspection device.

5.1.2 Camera - OV2460

OV2460 Datasheet

OV2460 camera which is widely available is chosen as the best suit because of the below pros.

- **High-Resolution Image Capture:**
  - The OV2460 camera is chosen for its fine resolution image capture capabilities, essential for detecting fine defects in PCBs.
  - It features a 2-megapixel sensor with an image resolution of 1600 x 1200 pixels, providing a fine image quality for inspection.
- **Optical Features:**
  - Low light sensitivity and high dynamic range ensure good performance in varied lighting conditions, which is crucial for inspecting PCBs where lighting conditions can vary significantly.
- **Digital Video Port (DVP) Interface:**
  - The OV2460 utilizes a digital video port (DVP) interface, making it compatible with the ESP32S microcontroller and ensuring smooth integration and performance.
- **Power Requirements:**
  - The camera requires 2.8V for its core operation and 1.2V for I/O, which can be supplied by the regulated power supply of the inspection device, simplifying power management and integration.

5.1.3 Power Supply

The power supply for the PCB inspection device is obtained from a micro USB port, providing a standard 5V supply.

- **Voltage Regulation:**
  - The 5V voltage from the micro USB port is regulated to meet the specific voltage requirements of the device components.
  - The microcontroller operates at 3.3V, while the camera requires 2.8V for its core operation and 1.2V for its I/O.
- **Stable Power Supply:**
  - Voltage regulators are employed to ensure a stable power supply, crucial for the reliable operation of both the microcontroller and the camera.
  - A stable power supply enhances the overall reliability and performance of the inspection device, preventing potential power-related issues.

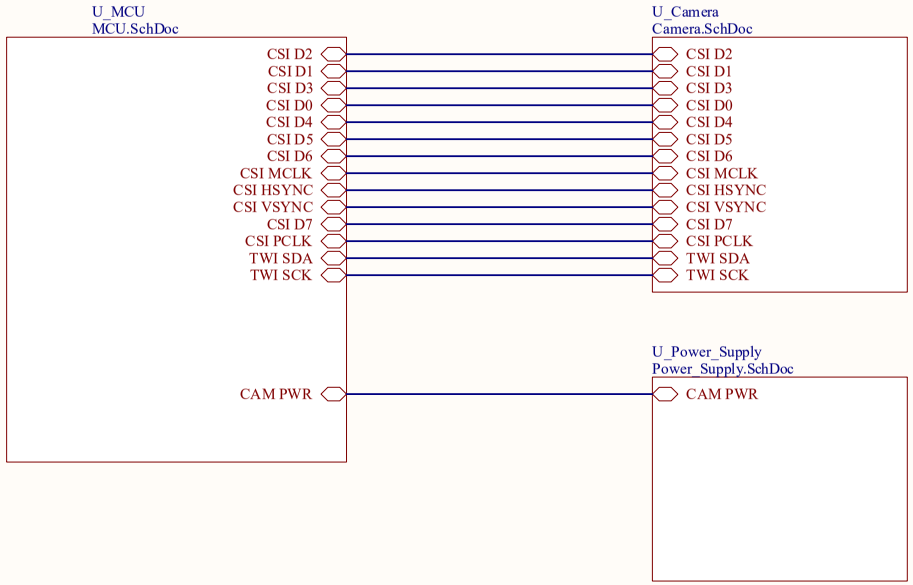
- **Convenience and Availability:**
  - Using a micro USB port as the power source offers convenience and widespread availability, making it easy to power the device in various settings.

5.1.4 Lighting System

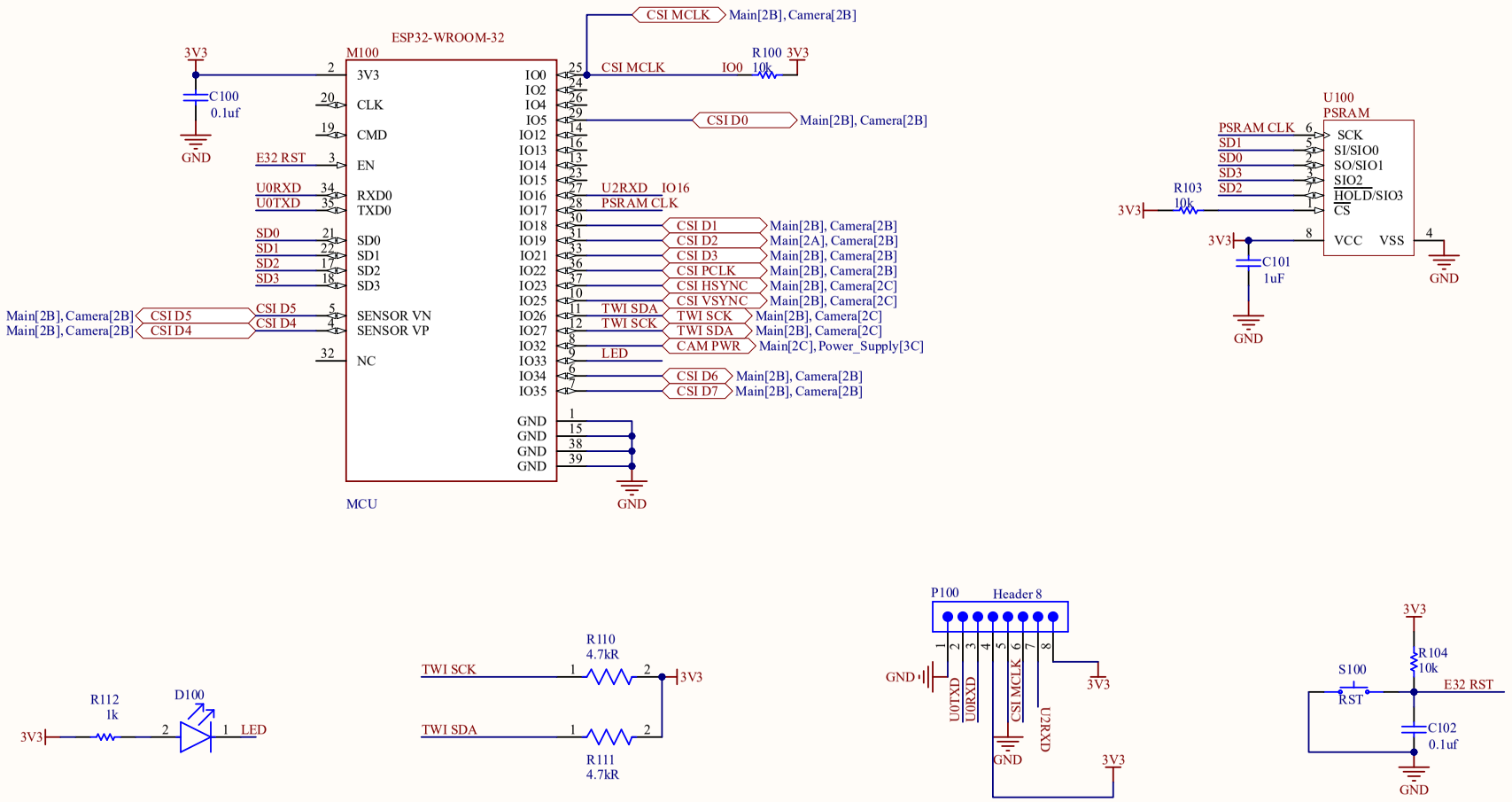
Proper lighting is essential for achieving high-quality images and accurate defect detection during PCB inspection. The power supply for the PCB inspection device is also obtained from the micro USB port, providing a standard 5V supply.

- **LED Strip:**
  - The lighting system utilizes a high-brightness LED strip, providing the necessary illumination for inspecting PCBs.
- **Voltage Regulation:**
  - The voltage for the LED strip is regulated by a potentiometer, allowing adjustment of the light intensity.
  - This feature is crucial for adapting to different inspection environments, ensuring optimal illumination under varying conditions.
- **Enhanced Versatility and Usability:**
  - By allowing the light intensity to be adjusted, the lighting system ensures that the inspection device can function effectively in diverse settings.
  - This enhances the device's versatility and usability, making it suitable for a wide range of inspection scenarios.

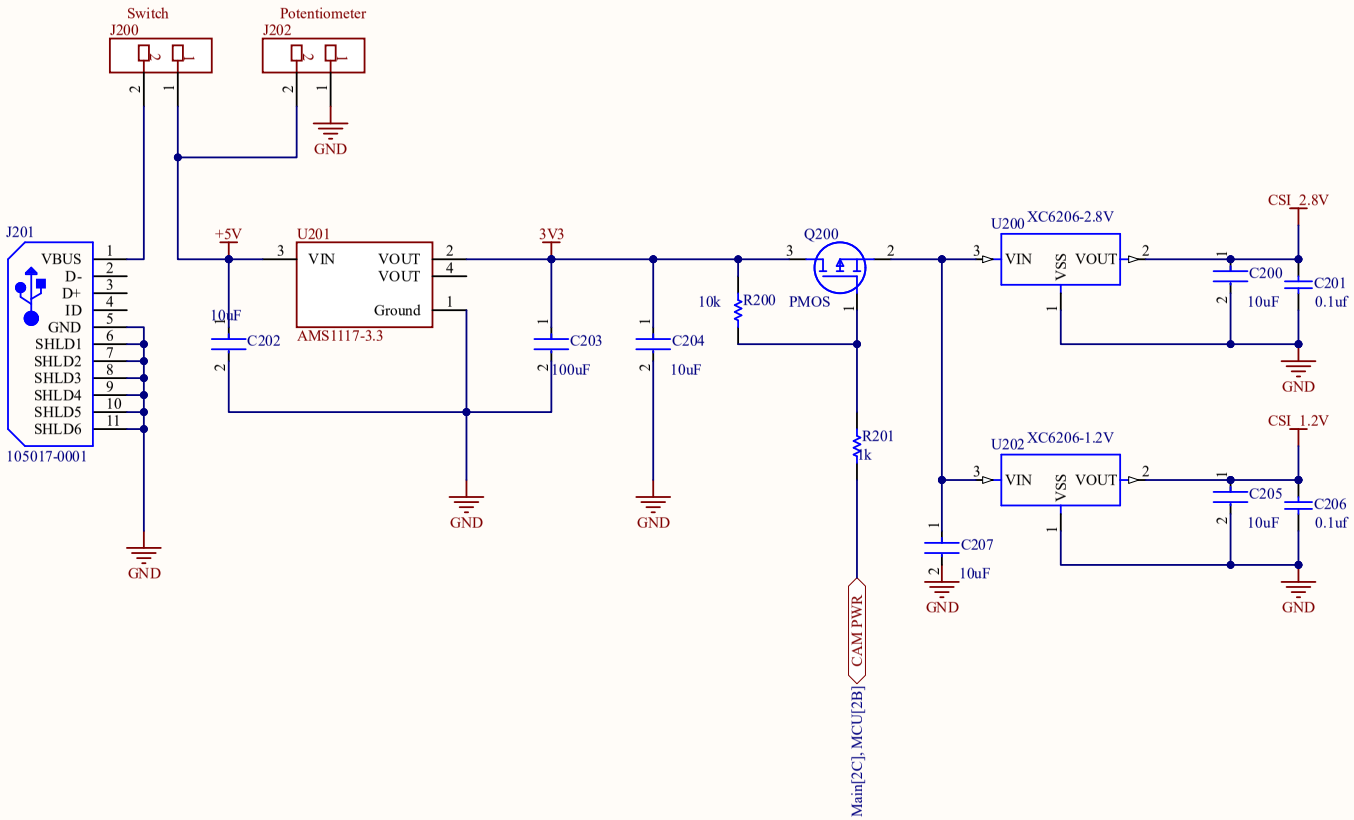
5.2 Schematic Diagram



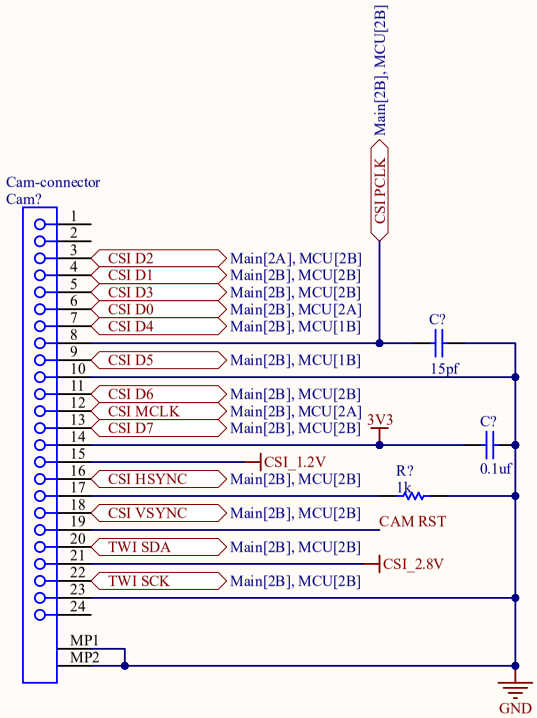
Title	Main		
Size: A4	Number: P1_210608	Revision: V1	
Date: 5/30/2024		Sheet bf 4	
File: D:\UoM\..\Main.SchDoc		Drawn By: L.J.J.P. Silva	

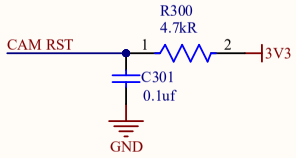


Title	MCU		
Size: A4	Number: P1_210608	Revision: V1	
Date: 5/30/2024	Sheet 2 of 4		
File: D:\PCBdefects\Altium\MCU.SchDoc	Drawn By: L.J.J.P. Silva		



Title	Power Supply	
Size: A4	Number: P1_210608	Revision: V1
Date: 5/30/2024	Sheet 3 of 4	
File: D:\UoM\..\Power_Supply.SchDoc	Drawn By: L.J.J.P. Silva	





Title	Camera		
Size: A4	Number: P1_210608	Revision: V1	
Date: 5/30/2024		Sheet 4 of 4	
File: D:\UoM\..\Camera Mod.SchDoc		Drawn By: L.J.J.P. Silva	

6 PCB Design - Version A 14/4/2024

6.1 Design Considerations

When designing the PCB the following crucial factors are taken into account.

1. **Compact PCB Layout**
    - The PCB should be as compact as possible to minimize material costs and improve the overall device's portability and aesthetic.
    - A compact design also reduces the signal path lengths, which can help in minimizing electromagnetic interference (EMI) and signal loss.
2. **ESP32 Placement and Antenna Positioning**
    - The ESP32 module is placed on an edge of the PCB with the antenna extending outside the PCB. This placement is critical to avoid signal interference from other components and to ensure optimal wireless communication performance.
    - The antenna needs to be free from obstructions to maintain a strong and clear signal for WiFi and Bluetooth connectivity.
3. **Power Input Micro USB Adapter Positioning**
    - The power input micro USB adapter is positioned in a corner of the PCB. Placing it in the corner allows for easier access and connection, especially when the device is housed in the enclosure.
    - It also helps in segregating the power input area from sensitive analog or RF sections of the PCB, reducing potential noise interference.
4. **Decoupling Capacitors Placement**
    - Decoupling capacitors are placed near the pins of the ESP32 and other ICs.
    - These capacitors help in filtering out noise and providing a stable power supply to the ICs by smoothing out voltage fluctuations.
    - Proper placement close to the pins is essential for their effectiveness in noise reduction and voltage stabilization.
5. **Camera Placement**
    - The camera module is placed on the bottom side of the PCB.
    - This decision is influenced by the mechanical design of the final product, where the camera needs to face outward through a designated opening.
6. **Hole Sizes for Mounting**
    - 3mm hole sizes are used for mounting, considering availability and size standards.

- These holes are crucial for securing the PCB within its housing or to other mechanical components.
- Standard hole sizes ensure compatibility with commonly used screws and standoffs, simplifying the assembly process.

7. **Ground Polygon Pours**
   - Ground polygon pours are used in both top and bottom layers of the PCB.
   - Ground pours help in reducing noise by providing a continuous ground plane, which is essential for signal integrity and EMI control.
   - They also aid in thermal management by distributing heat more evenly across the PCB.
8. **JST Connectors for Buttons and Potentiometer**
   - Two JST connectors are included for the button and potentiometer needed for the LED strip.

6.2 Trace Width Calculations

When designing a PCB, it is crucial to ensure that the trace widths are sufficient to handle the expected current without causing excessive heating.

Using the IPC-2221 standard formula for external layers and 1 oz copper (35 micrometers thickness), the required trace width W in millimeters for a given current I and temperature rise T_r is:

$$W = \frac{I}{0.048 \cdot (T_r)^{0.44}}$$

Where:

- $T_r = 10°\text{C}$
- $(T_r)^{0.44} \approx 2.88$
- For external layers, $k = 0.048$

6.2.1 Required Current calculation for the Components

- **ESP32 chip and Camera (3.3V supply):** Typically draws around 0.5A during operation.
- **LED Strip (5V supply):** Assuming each LED consumes about 60mA, a strip of 20 LEDs will draw:

$$20 \text{ LEDs} \times 60 \text{ mA} = 1200 \text{ mA} = 1.2 \text{ A}$$

Trace Width Calculations

For the 5V Trace

$$W_{5V} = \frac{1.2}{0.048 \cdot (10)^{0.44} \cdot (1)^{0.725}} \approx \frac{1.2}{0.048 \cdot 2.273 \cdot 1} \approx \frac{1.2}{0.109} \approx 11.01 \text{ mils}$$

Converting to millimeters:

$$W_{5V} \approx 0.279 \text{ mm}$$

For the 3.3V Trace

$$W_{3.3V} = \frac{0.5}{0.048 \cdot (10)^{0.44} \cdot (1)^{0.725}} \approx \frac{0.5}{0.048 \cdot 2.273 \cdot 1} \approx \frac{0.5}{0.109} \approx 4.59 \text{ mils}$$

Converting to millimeters:

$$W_{3.3V} \approx 0.117 \text{ mm}$$

For Signal Traces:

Signal traces typically carry very low currents, usually in the milliampere range (e.g., 10mA). For such low currents, a standard minimum trace width is generally used for manufacturing tolerances and reliability, typically around 6 mils (0.152 mm).

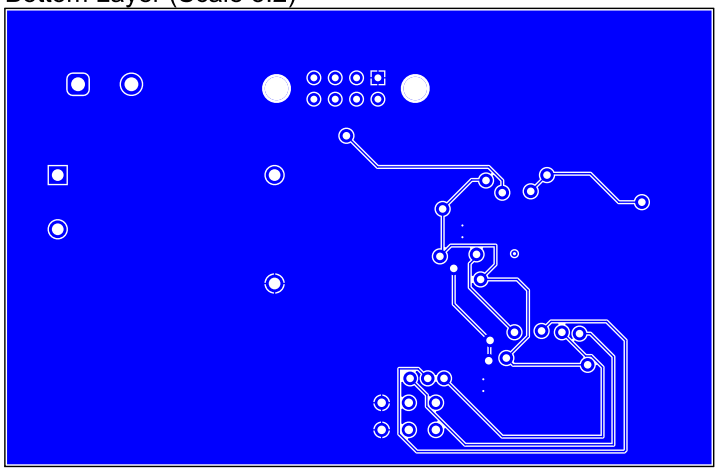
Summary of Trace Widths

Buy keeping an additional margin the trace widths were set as below.

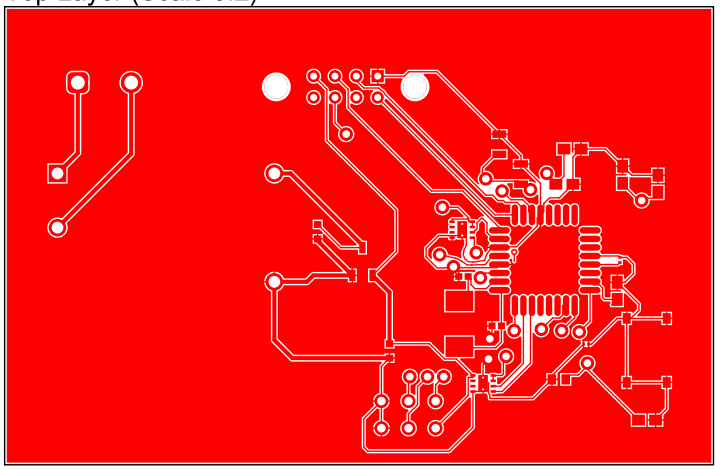
- **5V Trace (to LED Strip):** 1 mm
- **3.3V Trace (to ESP32):** 0.7 mm
- **Signal Traces:** 0.254 mm

6.3 Final PCB Design

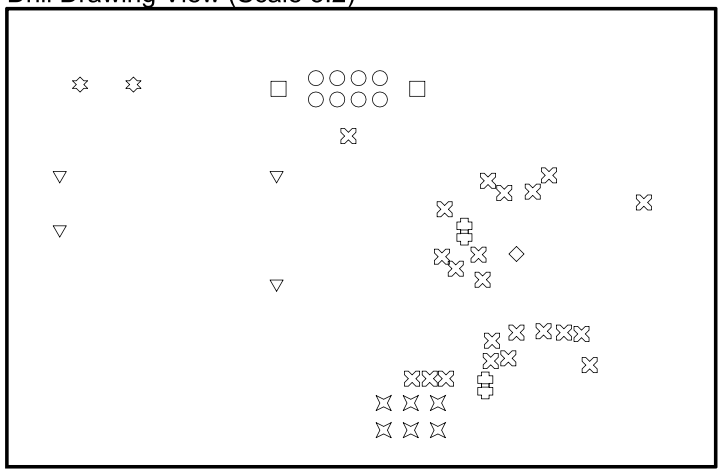
Bottom Layer (Scale 3:2)



Top Layer (Scale 3:2)



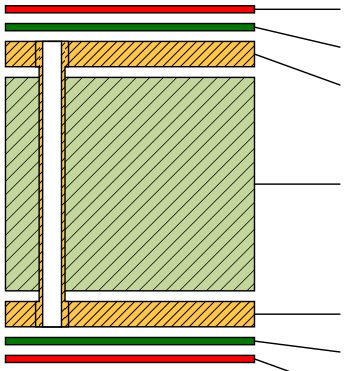
Drill Drawing View (Scale 3:2)



Drill Table

Symbol	Count	Hole Size	Plated	Hole Tolerance
⌖	4	0.20mm	Plated	
◇	1	0.30mm	Plated	
○	8	0.65mm	Plated	
⊠	22	0.71mm	Plated	
⊠	6	0.80mm	Plated	
▽	4	1.10mm	Plated	
✡	2	1.30mm	Plated	
□	2	2.20mm	Non-Plated	
	49 Total			

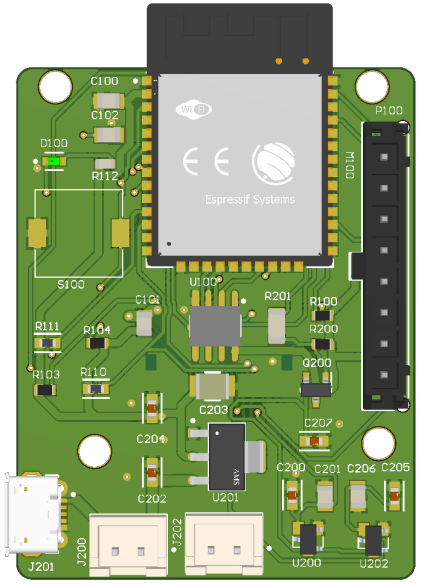
Layer Stack Legend



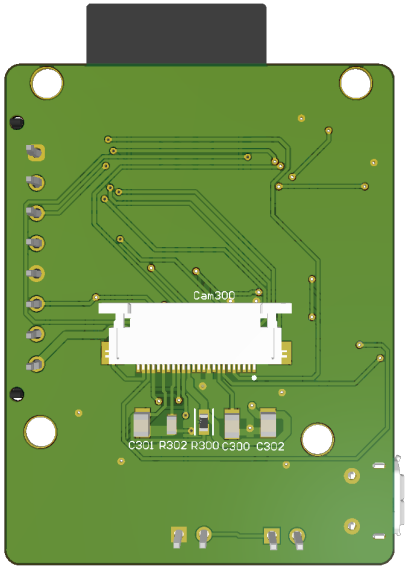
Material	Layer	Thickness	Dielectric Material	Type	Gerber
	Top Overlay			Legend	GTO
Surface Material	Top Solder	0.01mm	Solder Resist	Solder Mask	GTS
Copper	**Top Layer**	**0.04mm**		**Signal**	**GTL**
		0.32mm	*FR-4*	*Dielectric*	
Copper	**Bottom Layer**	**0.04mm**		**Signal**	**GBL**
Surface Material	Bottom Solder	0.01mm	Solder Resist	Solder Mask	GBS
	Bottom Overlay			Legend	GBO

Total thickness: 0.41mm

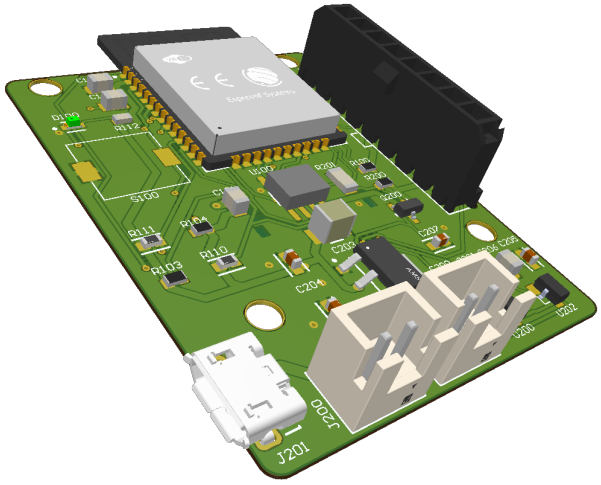
Realistic View- Top



Realistic View - Bottom



Realistic View -Side

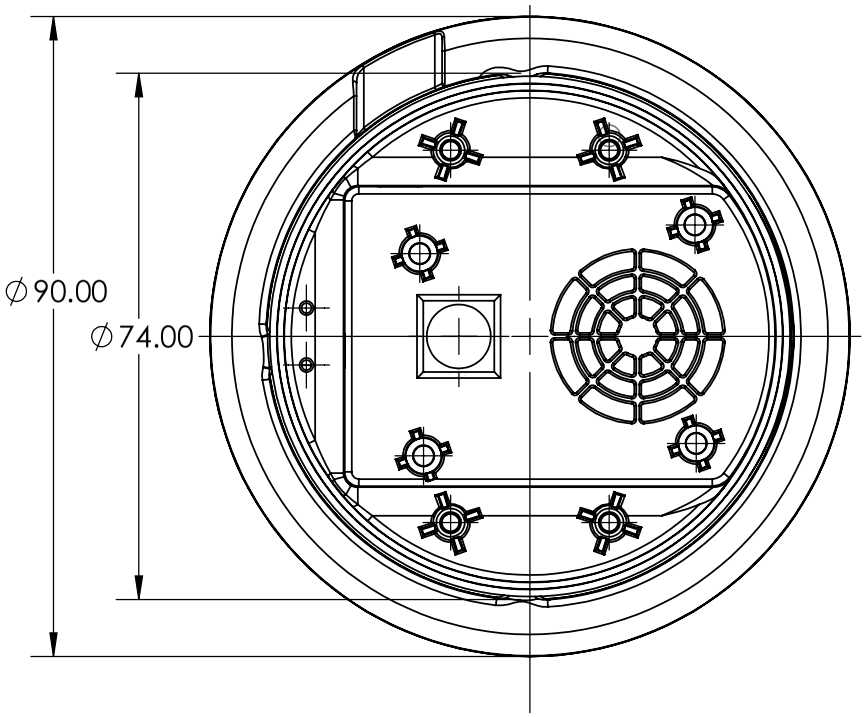


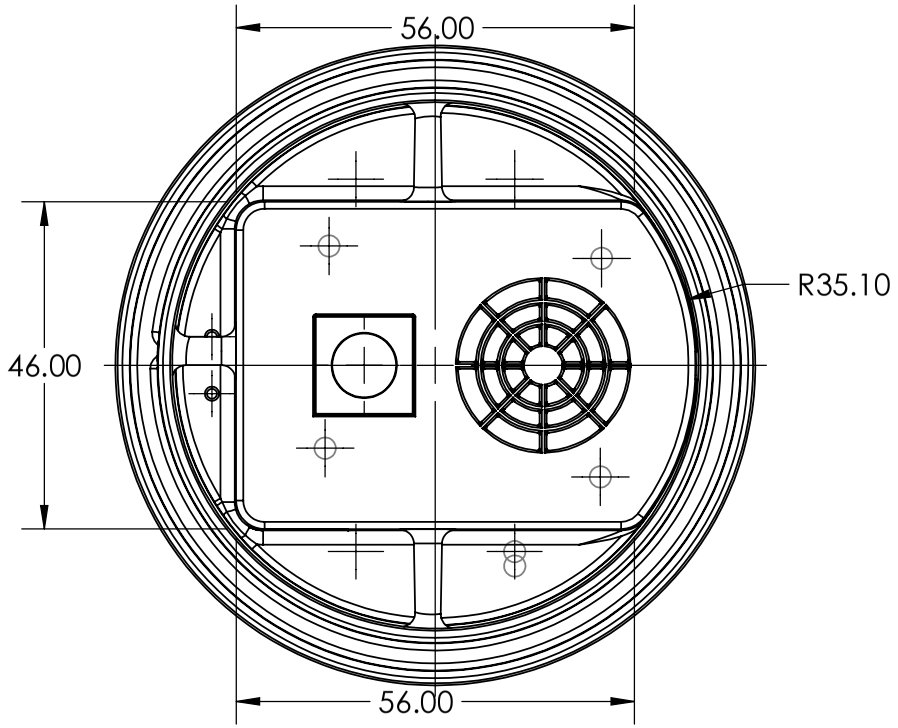
7 Enclosure Design - Version A 24/4/2024

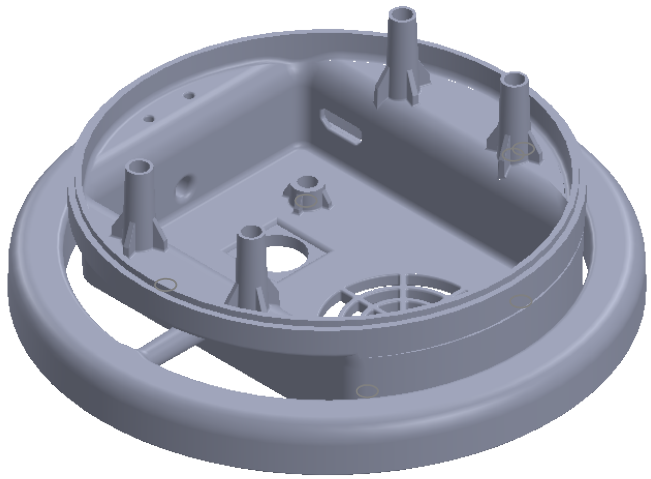
When designing the enclosure, the following factors were taken into account to ensure functionality, ease of use, and optimal performance of the integrated components:

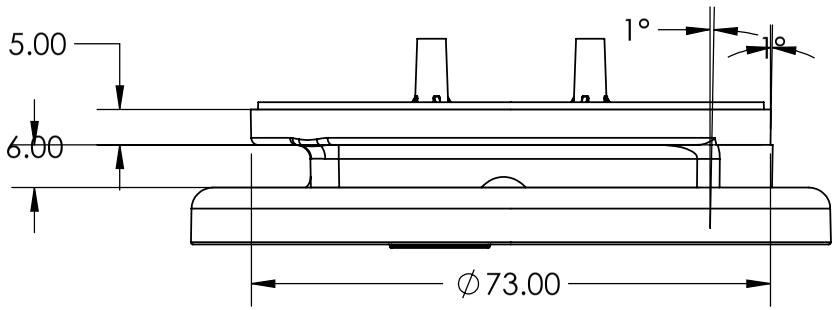
1. **Place to Mount the Camera and PCB**
    - The enclosure includes dedicated mounting points for the camera and the PCB.
    - This ensures that the camera is securely positioned to capture clear images of the PCB, and the PCB is held firmly in place for inspection and testing.
2. **Acrylic Flat Surface for PCB Inspection**
    - An acrylic flat surface is included in the design to provide a clear, stable platform for the PCB.
3. **Adjustable Height for Camera Projection**
    - An aluminum bar is used to adjust the height of the camera relative to the PCB.
    - This feature allows users to easily set the optimal distance for capturing detailed images, accommodating various sizes and types of PCBs.
4. **Ring of Light**
    - A ring of light is incorporated around the camera mount to ensure even and sufficient illumination of the PCB.
    - Proper lighting is crucial for clear imaging and accurate inspection, eliminating shadows and highlighting fine details.
5. **On/Off Button**
    - The enclosure includes an on/off button for controlling the power supply to the entire setup.
    - This adds convenience and safety, allowing users to easily power the system up or down as needed.

By incorporating these features, the enclosure design ensures a user-friendly and efficient setup for inspecting and working with PCBs.

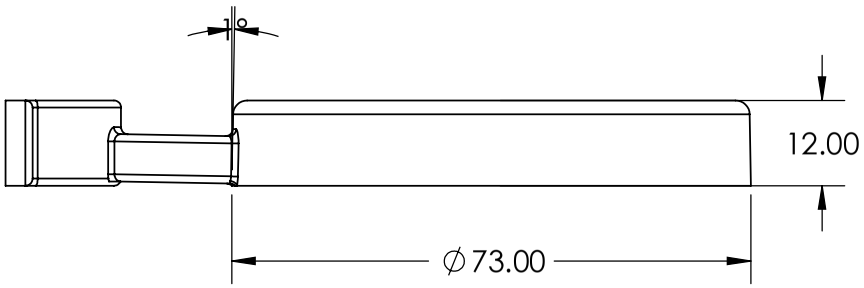


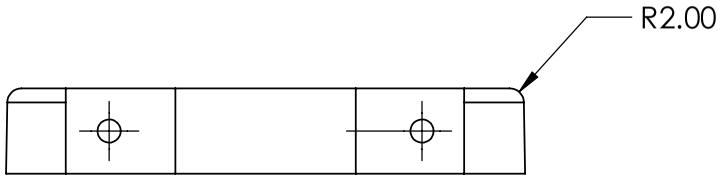


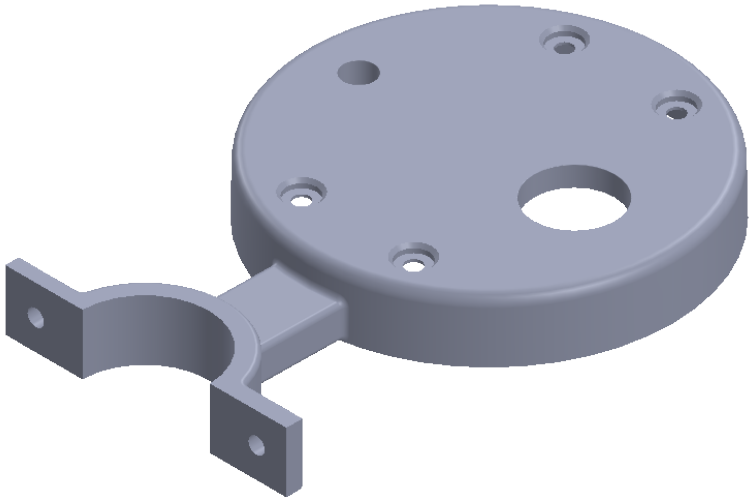


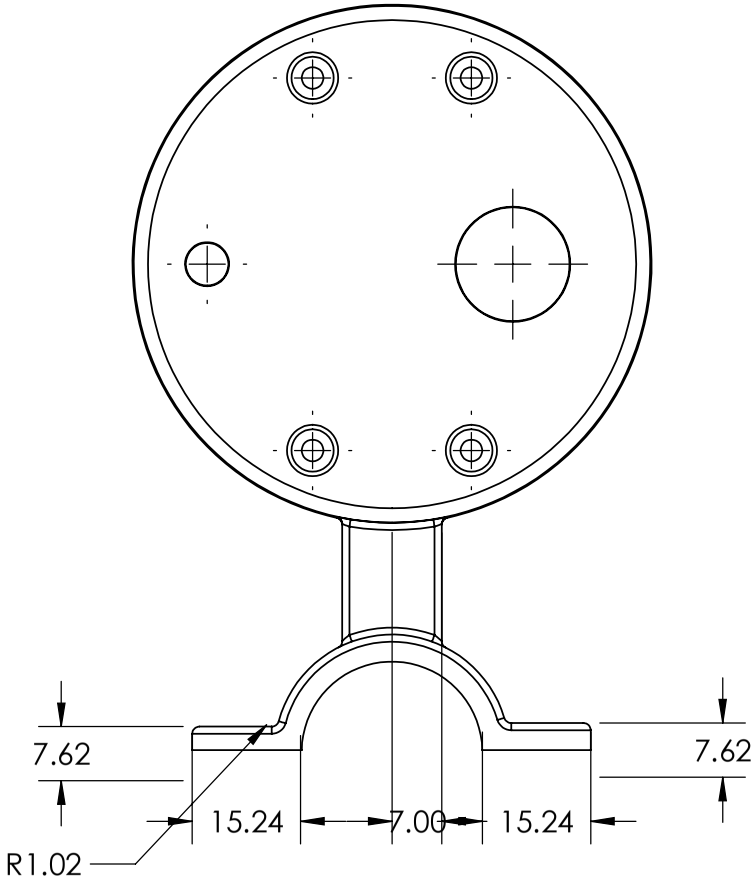


TITLE: PCB INSPECTION DEVICE				DO NOT SCALE DRAWING	REVISION V1
				UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES	
	NAME	DATE	MATERIAL:	DWG NO. CAM_BOTTOM	A4
DRAWN	SILVA L.J.J.P	15/04/2024	THERMOPLASTIC POLYMERS		
CHK'D	NAWARATHNE B.	20/04/2024	WEIGHT:	SCALE:1:1	SHEET 1 OF 3

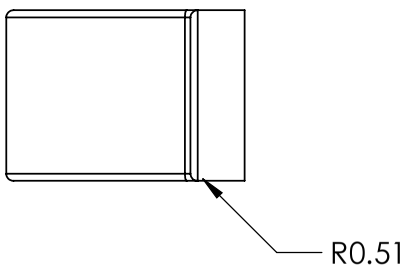


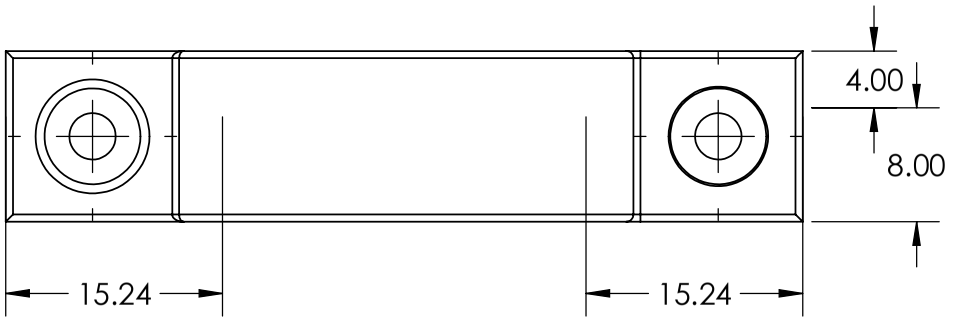


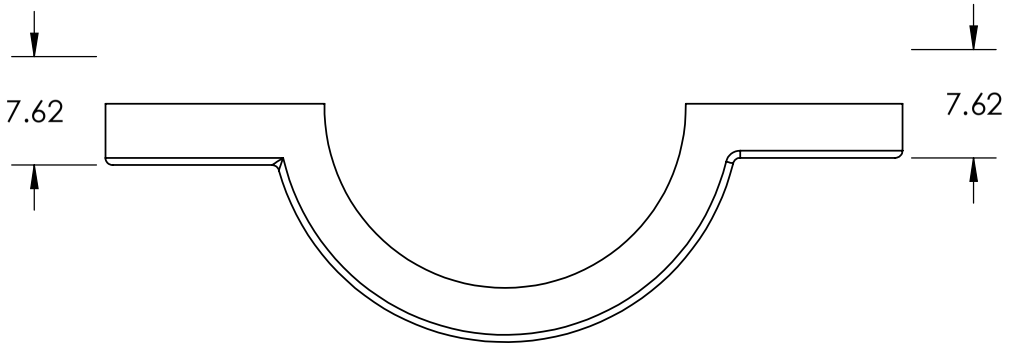


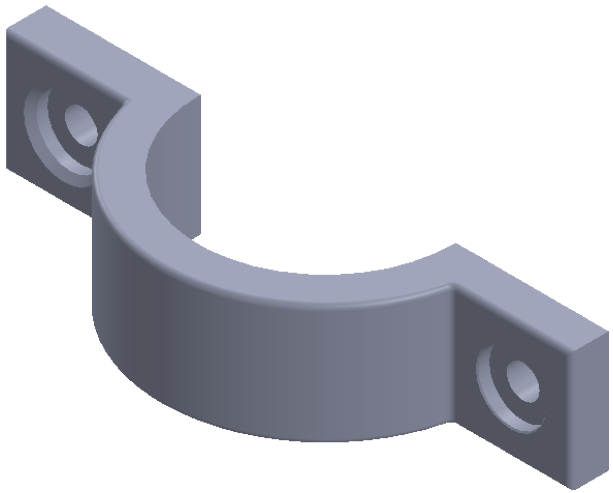


TITLE: PCB INSPECTION DEVICE				DO NOT SCALE DRAWING	REVISION V1
				UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS	
	NAME	DATE	MATERIAL:	DWG NO.	A4
DRAWN	SILVA L.J.J.P	15/04/2024	THERMOPLASTIC POLYMERS	CAM_TOP	
CHK'D	NAWARATHNE B.	20/04/2024	WEIGHT:	SCALE:1:2	SHEET 2 OF 3

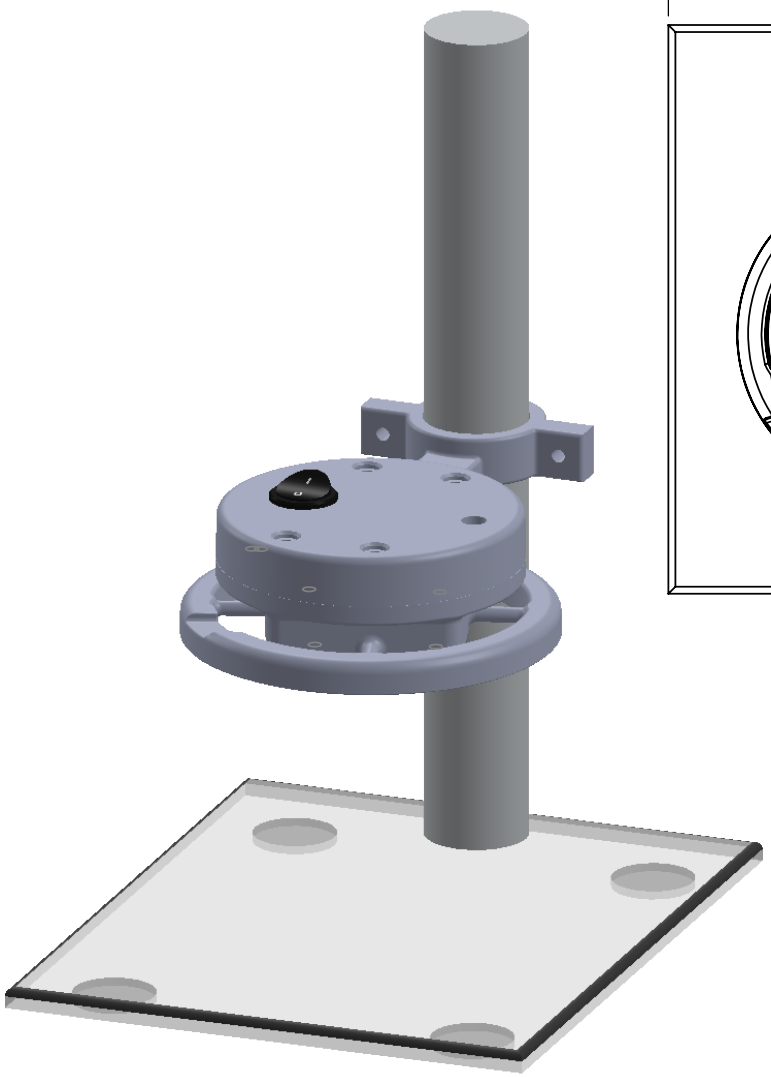


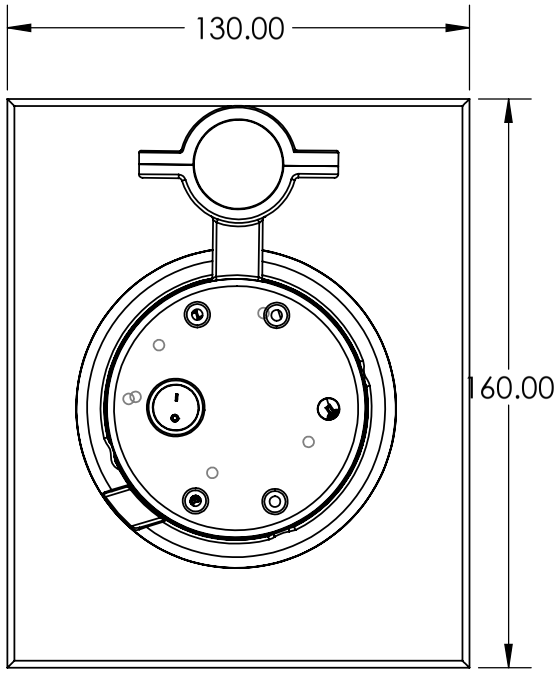


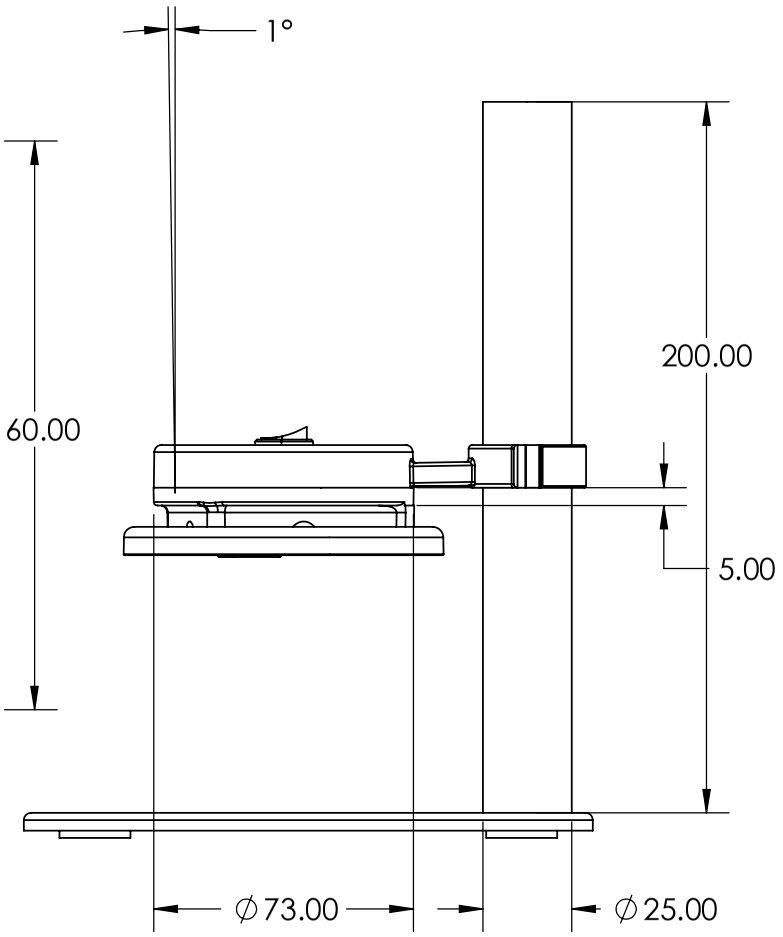




TITLE: PCB INSPECTION DEVICE				DO NOT SCALE DRAWING	REVISION V1
				UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS	
	NAME	DATE	MATERIAL:	DWG NO.	A4
DRAWN	SILVA L.J.J.P	15/04/2024	THERMOPLASTIC POLYMERS	CAM_JOINT	
CHK'D	NAWARATHNE B.	20/04/2024	WEIGHT:	SCALE:2:1	SHEET 3 OF 3







TITLE: PCB INSPECTION DEVICE				DO NOT SCALE DRAWING	REVISION V1
				UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS	
	NAME	DATE	MATERIAL: THERMOPLASTIC POLYMERS ALUMINIUM ACRYLIC	DWG NO. FINAL_ASSEMBLY	A4
DRAWN	SILVA L.J.J.P	17/04/2024			
CHK'D	NAWARATHNE B.	20/04/2024	WEIGHT:	SCALE:1:4	SHEET 1 OF 1

8 Programme the ESP32

The ESP32 is programmed with Arduino IDE, also a FTDI programmer is needed to connect it to the computer and upload code.

8.1 Bill of materials

S.N.	Components	Quantity
1	FTDI Module	1
2	Micro-USB Cable	1
3	Jumper Wires	10

Table 1: List of Components

8.2 FTDI Connection 5/5/2024

The board doesn't have a programmer chip. So In order to program, any type of USB-to-TTL Module can be used. There are so many FTDI Module available based on CP2102 or CP2104 Chip or any other chip.

Make a following connection between FTDI Module and ESP32 CAM module.

ESP32	FTDI Programmer
GND	GND
5V	VCC
U0R	TX
U0T	RX
GPIO0	GND

Table 2: ESP32 to FTDI Programmer Connections

Connect the **5V & GND** Pin of ESP32 to 5V & GND of FTDI Module. Similarly, connect the Rx to UOT and Tx to UOR Pin. And the most important thing, **you need to short the IO0 and GND Pin together**. This is to **put the device in programming mode**. Once programming is done you can remove it.

8.3 Installing ESP32CAM Library

Here we will **not use** the general **ESP webserver** example rather another streaming process. Therefore we need to add another ESPCAM library. The esp32cam library provides an object oriented API to use OV2640 camera on ESP32 microcontroller. It is a wrapper of esp32-camera library.

Go to the following Github Link and download the zip library as in the image.

Once downloaded add this zip library to Arduino Libray Folder. To do so follow the following steps:

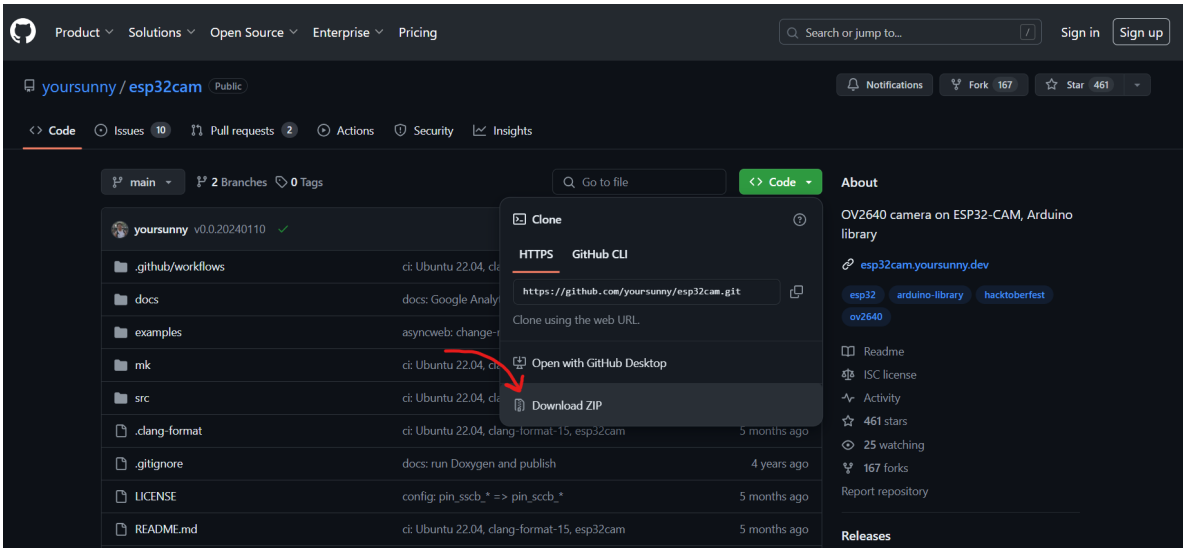


Figure 10: Github Page to Download the Library

Open **Arduino** → **Sketch** → **Include Library** → **Add .ZIP Library...** → Navigate to downloaded zip file → **add**

8.4 Source Code

Here is a source code. Copy the code and paste it in the Arduino IDE.

```
#include <WebServer.h>
#include <WiFi.h>
#include <esp32cam.h>

const char* WIFI_SSID = "ssid";
const char* WIFI_PASS = "password";

WebServer server(80);


static auto loRes = esp32cam::Resolution::find(320, 240);
static auto midRes = esp32cam::Resolution::find(350, 530);
static auto hiRes = esp32cam::Resolution::find(800, 600);
void serveJpg()
{
  auto frame = esp32cam::capture();
  if (frame == nullptr) {
    Serial.println("CAPTURE FAIL");
    server.send(503, "", "");
    return;
  }
  Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->
```

```
    ↪ getHeight(),
                static_cast<int>(frame->size()));

  server.setContentLength(frame->size());
  server.send(200, "image/jpeg");
  WiFiClient client = server.client();
  frame->writeTo(client);
}

void handleJpgLo()
{
  if (!esp32cam:: Camera.changeResolution(loRes)) {
    Serial.println("SET-LO-RES FAIL");
  }
  serveJpg();
}

void handleJpgHi()
{
  if (!esp32cam:: Camera.changeResolution(hiRes)) {
    Serial.println("SET-HI-RES FAIL");
  }
  serveJpg();
}

void handleJpgMid()
{
  if (!esp32cam::Camera.changeResolution(midRes)) {
    Serial.println("SET-MID-RES FAIL");
  }
  serveJpg();
}


void  setup(){
  Serial.begin(115200);
  Serial.println();
  {
    using namespace esp32cam;
    Config cfg;
    cfg.setPins(pins::AiThinker);
    cfg.setResolution(hiRes);
    cfg.setBufferCount(2);
    cfg.setJpeg(80);

    bool ok =  Camera.begin(cfg);
    Serial.println(ok ? " CAMERA OK" : " CAMERA FAIL");
```

```
  }
  WiFi.persistent(false);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  Serial.print("http://");
  Serial.println(WiFi.localIP());
  Serial.println("  /cam-lo.jpg");
  Serial.println("  /cam-hi.jpg");
  Serial.println("  /cam-mid.jpg");

  server.on("/cam-lo.jpg", handleJpgLo);
  server.on("/cam-hi.jpg", handleJpgHi);
  server.on("/cam-mid.jpg", handleJpgMid);

  server.begin();
}

void loop()
{
  server.handleClient();
}
```

Before uploading the code, you have to make a small change to the code. Change the `SSID` and `password` variables in accordance with your WiFi network.

Now compile and upload it to the ESP32. During uploading, you have to follow a few steps every time:

1. Make sure the `IO0` pin is shorted with the ground when you press the upload button.
2. If you see dots and dashes while uploading, press the reset button immediately.
3. Once the code is uploaded, remove the `IO0` pin shorting with Ground and press the reset button once again.
4. If the output in the Serial Monitor is still not there, then press the reset button again.

Now you can see a similar output as in the image below.

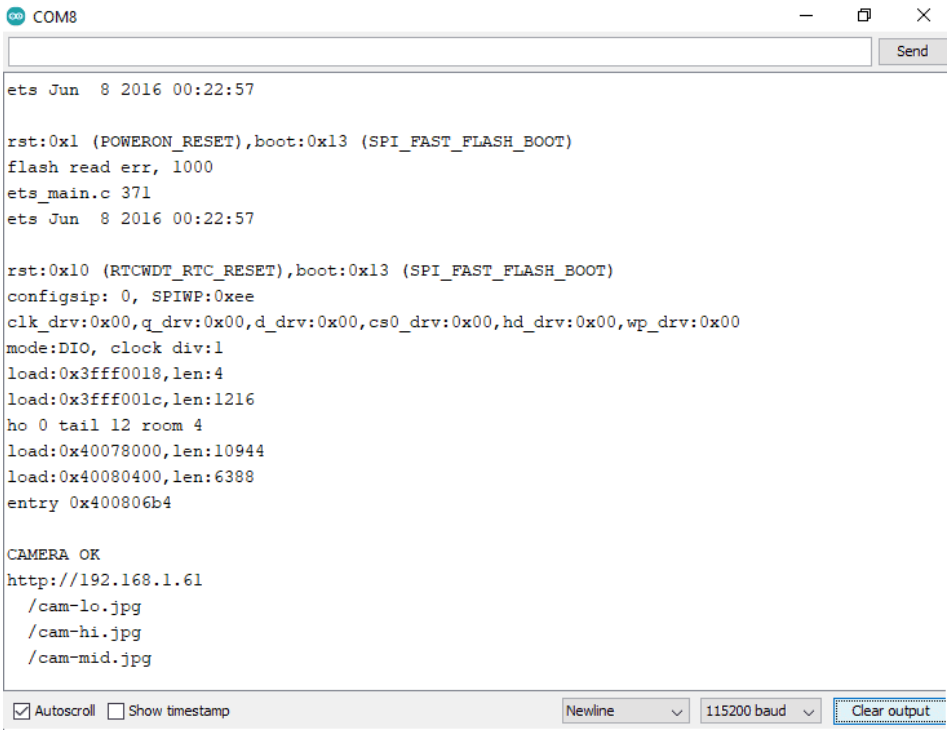


Figure 11: Seriel Output

9 Building the Web Application

9.1 Choosing Frameworks 10/5/2024

Why Streamlit?

Streamlit is chosen as the framework to build the web application due to its simplicity and ease of use in creating interactive web applications especially related to AI. It allows for rapid development and deployment of data applications with minimal code. Key features of Streamlit that make it suitable for this project include:

- **Interactive Widgets**: Streamlit provides a wide range of widgets to create interactive elements such as buttons, sliders, and file uploaders.
- **Real-time Updates**: Streamlit's ability to update the web interface in real-time is crucial for applications like live PCB inspection.
- **Ease of Integration**: Streamlit easily integrates with popular data science libraries such as OpenCV, PIL, and YOLO, facilitating the development of complex applications.
- **Rapid Prototyping**: The straightforward syntax and automatic reloading capabilities allow for quick iteration and testing of the application.

9.2 Application Structure

The PCB Inspection application offers a user-friendly interface designed for detecting defects in printed circuit boards (PCBs). The app is built enabling users to either

perform real-time inspections via a live camera feed or upload images for analysis. Below is a detailed report on the various pages and functionalities of the application.

9.2.1 Home Page

The home page greets users with a large title: **"Welcome to PCB Inspector!"**. A markdown section provides an overview of the application, explaining its purpose and the types of PCB defects it can detect, including:

- Missing Hole
- Mouse Bite
- Open Circuit
- Short Circuit
- Spur
- Spurious Copper

Users are informed about two inspection options: **Live Inspection** and **Upload Images**.

9.2.2 Sidebar Menu

The sidebar allows users to navigate between different modes of operation:

- About
- PCB Defects
- Component Defects

9.2.3 PCB Defects Page

Model Configuration

The sidebar contains settings for configuring the machine learning model, including:

- Selecting the task: **Live Camera** or **Upload PCB Image**
- Adjusting the model confidence level through a slider.

Live Camera Mode

When **Live Camera** is selected, the application fetches continuous images from a specified URL. Users can process images by clicking the **Process Image** button, which triggers the `process_image` function. This function:

- Retrieves the current image.

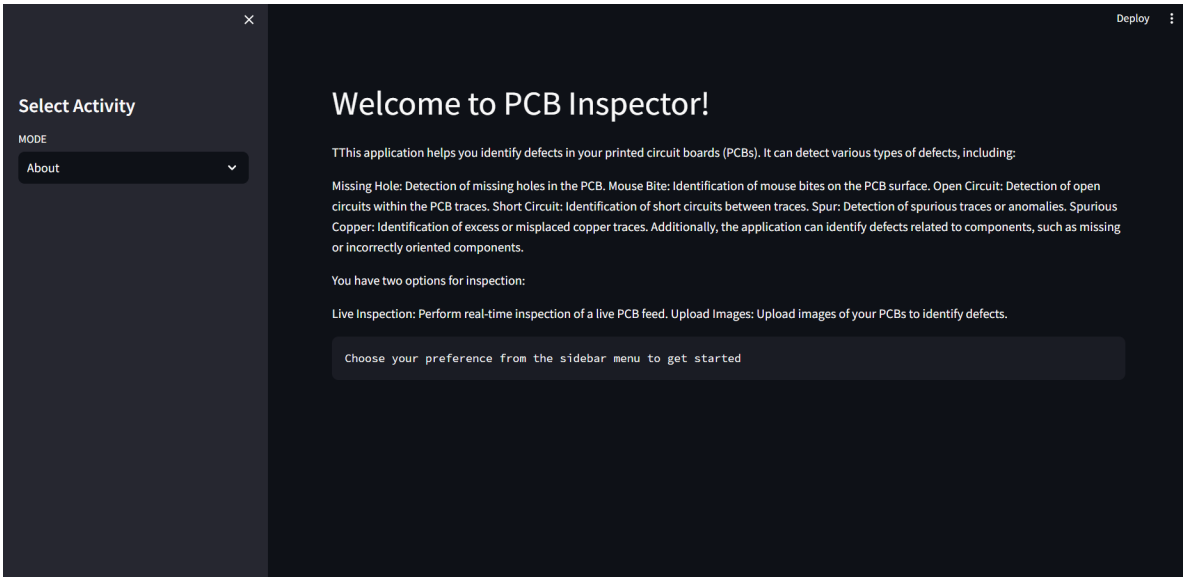


Figure 12: Home Page

Figure 13: PCB Defects tested by live camera

- Processes it using the YOLO model to detect defects.
- Displays the processed image with detected defects highlighted.

Upload PCB Image Mode

Users can upload images in various formats (jpg, jpeg, png, bmp, webp). The sidebar provides options to select defect types to predict. Upon clicking **Detect Objects**, the uploaded image is processed, and the detected defects are displayed.

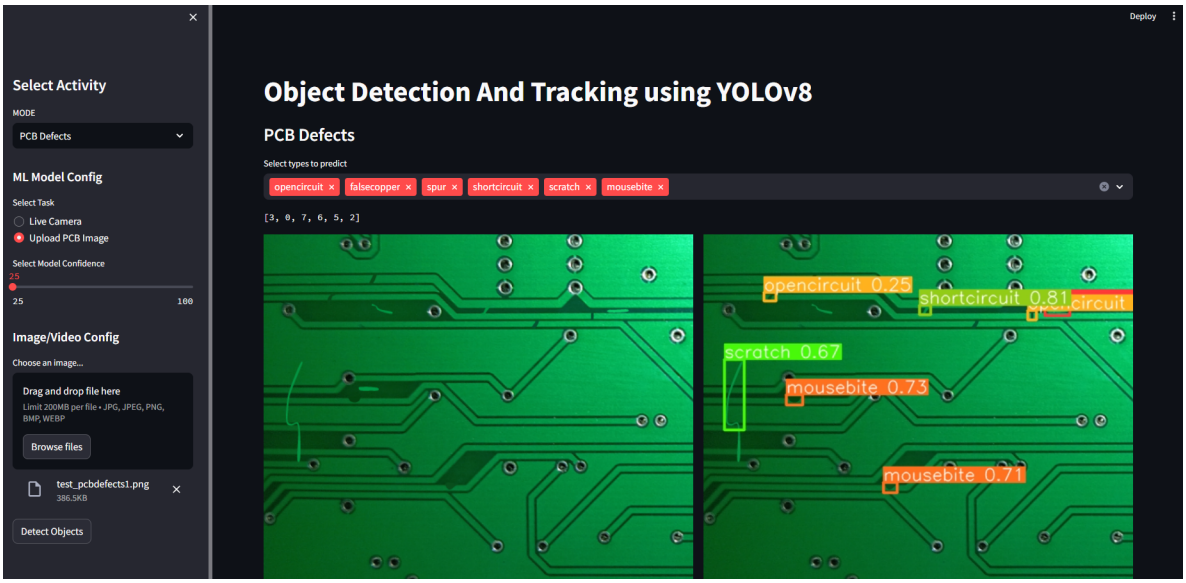


Figure 14: PCB Defects tested on Uploaded Image

9.2.4 Component Defects Page

Model Configuration

Similar to the PCB Defects page, users can choose between **Live Camera** and **Upload PCB Image** modes for inspecting component placement.

Live Camera Mode

Allows users to set a reference image and then process live images to detect discrepancies. The process involves:

- Capturing a reference image.
- Capturing and processing the current live image to find differences.
- Highlighting the differing regions to indicate potential defects.

Upload PCB Image Mode

Users upload a reference image and a source image for comparison. The application processes the source image against the reference to highlight differences, indicating potential component defects.

Figure 15: Components Defects tested by Live Camera

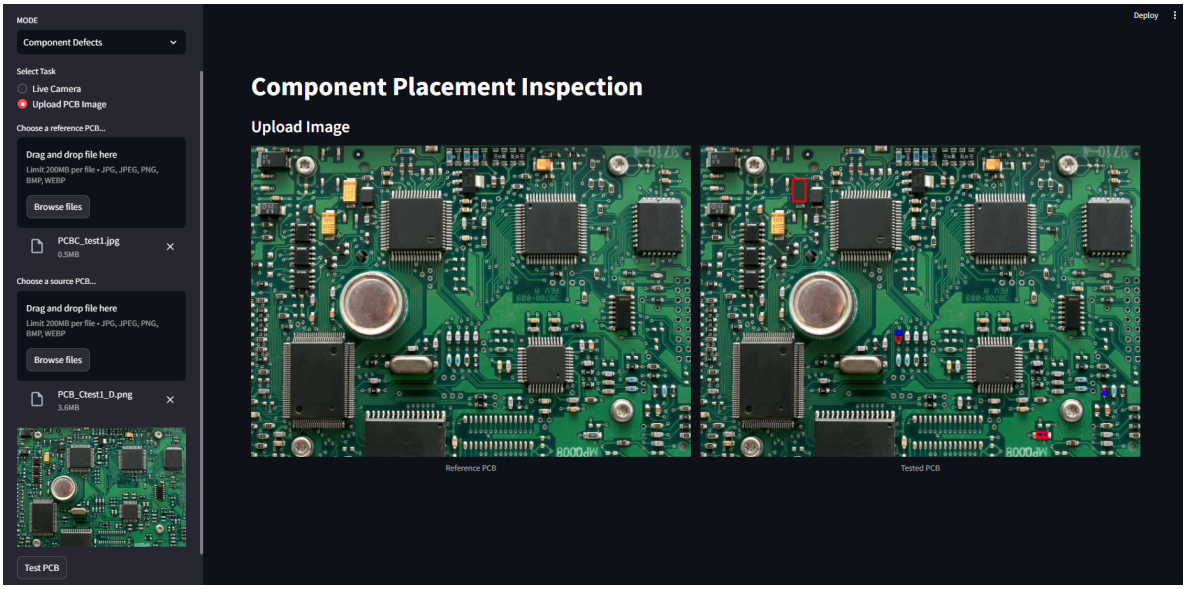


Figure 16: Components Defects tested on the uploaded image

9.3 Code Explanation

9.3.1 Importing Packages

The following Python packages are imported to support the functionalities of the application:

```
from ultralytics import YOLO
import cv2
import numpy as np
import streamlit as st
from PIL import Image
import requests
import urllib.request
from io import BytesIO
import time
```

- **YOLO from ultralytics**: This is the core object detection model used to identify defects in PCBs. YOLO is chosen for its high speed and accuracy.
- **cv2 (OpenCV)**: Used for various image processing tasks such as color conversion and contour detection.
- **numpy**: Used for numerical operations and array manipulations.
- **streamlit**: The web framework chosen for creating the interactive user interface.
- **PIL (Python Imaging Library)**: Used for opening and processing images.
- **requests and urllib**: Used for fetching images from a URL, particularly useful for live camera feeds.
- **BytesIO and time**: Used for handling byte streams and adding delays in the application.

9.3.2 Setting Up the Page Layout

The page layout of the Streamlit application is configured using the following code:

```
st.set_page_config(
    page_title="PCB Inspection",
    layout="wide",
    initial_sidebar_state="expanded"
)
```

9.3.3 Main Function and Sidebar Menu

The main function sets up the initial UI components and sidebar menu. Users can choose between "About," "PCB Defects," and "Component Defects."

```
def main():
    new_title = '<p style="font-size: 42px;">Welcome to PCB Inspector
    ↪ !</p>'
    read_me_0 = st.markdown(new_title, unsafe_allow_html=True)

    read_me = st.markdown("""
    This application helps you identify defects in your printed circuit
    ↪  boards (PCBs)...
    """)

    st.sidebar.title("Select Activity")
    choice = st.sidebar.selectbox("MODE", ("About", "PCB Defects", "
    ↪ Component Defects"))

    if choice == "PCB Defects":
        read_me_0.empty()
        read_me.empty()
        object_detection_image()
    elif choice == "Component Defects":
        read_me_0.empty()
        read_me.empty()
        Component_Defects()
    elif choice == "About":
        print()
```

9.3.4 Object Detection for PCB Defects

9.3.5 Live Camera Feed

The live camera feed function fetches images continuously from a specified URL, displays them in real-time, and processes the images when the "Process Image" button is clicked.

```
def object_detection_live():
    url = 'http://192.168.182.214/cam-hi.jpg'
    image_placeholder = st.empty()
    button_clicked = False

    process_button = st.button('Process Image')

    while True:
        if process_button:
            button_clicked = True
            process_button = False

        response = requests.get(url)
        image = Image.open(BytesIO(response.content))
```

```
        if button_clicked:
            process_image()
            button_clicked = False
            time.sleep(2)
        else:
            image_placeholder.image(image, channels="RGB")
        time.sleep(0.001)
```

9.3.6 Image Upload

This function allows users to upload an image, select specific defect types, and use the YOLO model to detect and highlight defects in the uploaded image.

```
def object_detection_upload():
    source_img = st.sidebar.file_uploader("Choose an image...", type=("
    ↪ jpg", "jpeg", "png", 'bmp', 'webp'))

    all_types = ['falsecopper','opencircuit','pinhole','scratch','
    ↪ shortcircuit','spur', 'missinghole', 'mousebite']
    selected_types = st.multiselect("Select types to predict",
    ↪ all_types)

    selected_indices = [get_index_by_value(type) for type in
    ↪ selected_types]

    st.text(selected_indices)

    col1, col2 = st.columns(2)

    with col1:
        try:
            if source_img is None:
                st.write("No image uploaded")
            else:
                uploaded_image = PIL.Image.open(source_img)
                st.image(source_img, caption="Uploaded Image",
    ↪ use_column_width=True)
        except Exception as ex:
            st.error("Error occurred while opening the image.")
            st.error(ex)

    with col2:
        if source_img is None:
            st.write("No image uploaded")
        else:
            if st.sidebar.button('Detect Objects'):
```

```
                res = model.predict(uploaded_image, classes=
    ↪ selected_indices)
                res_plotted = res[0].plot()[:, :, ::-1]
                st.image(res_plotted, caption='Detected Image',
    ↪ use_column_width=True)
```

9.3.7 Component Defects Inspection

9.3.8 Live Camera Inspection

This function fetches images from a URL, sets a reference image, and then processes live images to detect component defects by comparing them with the reference image.

```
def Component_Defects_live():
    st.subheader('Live')
    url = 'http://192.168.182.214/cam-hi.jpg'
    image_placeholder = st.empty()
    button_clicked = False
    Rbutton_clicked = False

    Reference_button = st.button('Set Reference Image')
    process_button = st.button('Process Image')

    col1, col2 = st.columns(2)

    while True:
        with col1:
            if Reference_button:
                Rbutton_clicked = True
                Reference_button = False

            response = requests.get(url)
            Reference_image = Image.open(BytesIO(response.content))
            reference_img = cv2.cvtColor(np.array(Reference_image), cv2
    ↪ .COLOR_RGB2BGR)

            if Rbutton_clicked:
                st.image(Reference_image, caption='Reference Image')
                Rbutton_clicked = False
                time.sleep(2)

        with col2:
            if process_button:
                button_clicked = True
                process_button = False

            response = requests.get(url)
            image = Image.open(BytesIO(response.content))
```

```
            if button_clicked:
                source_img = cv2.cvtColor(np.array(image), cv2.
    ↪ COLOR_RGB2BGR)
                gray1 = cv2.cvtColor(reference_img, cv2.COLOR_BGR2GRAY)
                gray2 = cv2.cvtColor(source_img, cv2.COLOR_BGR2GRAY)
                difference = cv2.absdiff(gray1, gray2)
                _, thresholded = cv2.threshold(difference, 10, 255, cv2
    ↪ .THRESH_BINARY)
                contours, _ = cv2.findContours(thresholded, cv2.
    ↪ RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
                for contour in contours:
                    x, y, w, h = cv2.boundingRect(contour)
                    cv2.rectangle(reference_img, (x, y), (x + w, y + h)
    ↪ , (255, 0, 0), 5)

                st.image(reference_img, caption='Component Defects')
                time.sleep(5)
            else:
                image_placeholder.image(image, channels="RGB")
        time.sleep(0.001)
```

9.4 Complete Code of the Web Application

```
# Python In-built packages

from ultralytics import YOLO
from ultralytics.models.yolo.detect import DetectionPredictor
import cv2
import numpy as np
import streamlit as st
from PIL import Image
import requests
import urllib.request
from pathlib import Path
import PIL
from io import BytesIO
import time

# External packages
#import streamlit as st

# Local Modules
#import settings
#import helper
```

```
# Setting page layout

confidence= 0.3
model = YOLO('best (1).pt')

st.set_page_config(
    page_title="PCB Inspection ",
    layout="wide",
    initial_sidebar_state="expanded"
)
def main():
    new_title = '<p style="font-size: 42px;">Welcome to PCB Inspector
    ↪ !</p>'
    read_me_0 = st.markdown(new_title, unsafe_allow_html=True)

    read_me = st.markdown("""
    TThis application helps you identify defects in your printed
    ↪ circuit boards (PCBs). It can detect various types of defects,
    ↪ including:

    Missing Hole: Detection of missing holes in the PCB.
    Mouse Bite: Identification of mouse bites on the PCB surface.
    Open Circuit: Detection of open circuits within the PCB traces.
    Short Circuit: Identification of short circuits between traces.
    Spur: Detection of spurious traces or anomalies.
    Spurious Copper: Identification of excess or misplaced copper
    ↪ traces.
    Additionally, the application can identify defects related to
    ↪ components, such as missing or incorrectly oriented components.

    You have two options for inspection:

    Live Inspection: Perform real-time inspection of a live PCB feed.
    Upload Images: Upload images of your PCBs to identify defects.


        Choose your preference from the sidebar menu to get started"""
    )
    st.sidebar.title("Select Activity")
    choice  = st.sidebar.selectbox("MODE",("About","PCB Defects","
    ↪ Component Defects"))
    #["Show Instruction","Landmark identification","Show the #source
    ↪ code", "About"]

    if choice == "PCB Defects":
        #st.subheader("PCB Defects")
```

```
        read_me_0.empty()
        read_me.empty()
            #st.title('Object Detection')
        object_detection_image()
    elif choice == "Component Defects":
        read_me_0.empty()
        read_me.empty()
            #object_detection_video.has_beenCalled = False
        #object_detection_video()
        Component_Defects()


    elif choice == "About":
        print()
    # Main page heading


def object_detection_image():
    global confidence
    st.title("PCB Defects Detection")
    st.subheader("PCB Defects")


    # Sidebar
    st.sidebar.header("ML Model Config")

    # Model Options
    model_type = st.sidebar.radio(
        "Select Task", ['Live Camera', 'Upload PCB Image'])

    confidence = float(st.sidebar.slider(
        "Select Model Confidence", 25, 100, 40)) / 100

    # Selecting Detection Or Segmentation

    model=YOLO('best (1).pt')


    st.sidebar.header("Image/Video Config")

    source_img = None
    # If image is selected

    if model_type == 'Live Camera':
         object_detection_live()

    else:
```

```
        object_detection_upload()

def Component_Defects():
    st.title('Component Placement Inspection')

    model_type = st.sidebar.radio(
        "Select Task", ['Live Camera', 'Upload PCB Image'])

    if model_type == 'Live Camera':
         Component_Defects_live()

    else:
        Component_Defects_upload()



def process_image():
    model = YOLO('best (1).pt')

    url = 'http://192.168.182.214/cam-hi.jpg'
    image=urllib.request.urlopen(url)
    image = Image.open(image)
    #st.image(image=image)
    #result= model(image)
    #names = result[0].names
    #probability = result[0].probs

    res = model.predict(image,conf=confidence)
    boxes = res[0].boxes
    res_plotted = res[0].plot()[:, :, ::-1]
    st.image(res_plotted, caption='Detected Image',use_column_width=
   ↪ True)


    # Perform your image processing here using your YOLO model
    # Replace this placeholder with your actual image processing code
    #result = image  # Placeholder for the result
    return

def object_detection_live():
    #st.title('PCB Defects Detection')

    # URL to fetch continuous images
    url = 'http://192.168.182.214/cam-hi.jpg'
    image_placeholder = st.empty()  # Placeholder to update the image
    button_clicked = False
```

```
    # Create the button to process image
    process_button = st.button('Process Image')

    while True:
        # If the button is clicked, process the image
        if process_button:
            button_clicked = True
            process_button = False

        response = requests.get(url)
        image = Image.open(BytesIO(response.content))

        if button_clicked:
            process_image()

            button_clicked = False
            time.sleep(2)

        else:
            # Display the image without processing
            image_placeholder.image(image, channels="RGB")

        # Add a small delay to control the frame rate
        time.sleep(0.001)  # 0.1 second delay


def get_index_by_value(value):
    for key, val in model.names.items():
        if val == value:
            return key
    # If the value is not found, return None or any other indication as
    ↪  per your requirement
    return None

def object_detection_upload():
    source_img = st.sidebar.file_uploader(
            "Choose an image...", type=("jpg", "jpeg", "png", 'bmp', '
    ↪ webp'))

    all_types = ['falsecopper','opencircuit','pinhole','scratch','
    ↪ shortcircuit','spur', 'missinghole', 'mousebite']
    selected_types = st.multiselect("Select types to predict",
    ↪ all_types)

    selected_indices = [get_index_by_value(type) for type in
    ↪ selected_types]
```

```
    st.text(selected_indices)

    col1, col2 = st.columns(2)

    with col1:
        try:
            if source_img is None:
                    st.write("no image uploaded")
            else:
                uploaded_image = PIL.Image.open(source_img)
                st.image(source_img, caption="Uploaded Image",
                            use_column_width=True)
        except Exception as ex:
            st.error("Error occurred while opening the image.")
            st.error(ex)

    with col2:
        if source_img is None:
                st.write("no image uploaded")

        else:
            if st.sidebar.button('Detect Objects'):
                res = model.predict(uploaded_image,classes=
    ↪ selected_indices)
                boxes = res[0].boxes
                res_plotted = res[0].plot()[:, :, ::-1]
                st.image(res_plotted, caption='Detected Image',
                                use_column_width=True)

def Component_Defects_upload():
    st.subheader('Upload Image')

    reference_img = st.sidebar.file_uploader(
            "Choose a reference PCB...", type=("jpg", "jpeg", "png", '
    ↪ bmp', 'webp'))

    source_img = st.sidebar.file_uploader(
            "Choose a source PCB...", type=("jpg", "jpeg", "png", 'bmp'
    ↪ , 'webp'))
    if source_img is not None:
        st.sidebar.image(source_img)

    if reference_img is not None and source_img is not None:
        reference_img = Image.open(reference_img)
        source_img = Image.open(source_img)
```

```
col1, col2 = st.columns(2)

with col1:
    try:
        if reference_img is None:
                st.write("No Reference PCB uploaded")
        else:
            #uploaded_image = PIL.Image.open(source_img)
            st.image(reference_img, caption="Reference PCB",
                        use_column_width=True)
    except Exception as ex:
        st.error("Error occurred while opening the image.")
        st.error(ex)

with col2:
    if source_img is None:
            st.write("No testing PCB uploaded")

    else:
        if st.sidebar.button('Test PCB'):
            reference_img = cv2.cvtColor(np.array(reference_img),
↪ cv2.COLOR_RGB2BGR)
            source_img = cv2.cvtColor(np.array(source_img), cv2.
↪ COLOR_RGB2BGR)

            # Convert images to grayscale
            gray1 = cv2.cvtColor(reference_img, cv2.COLOR_BGR2GRAY)
            gray2 = cv2.cvtColor(source_img, cv2.COLOR_BGR2GRAY)
            # Compute absolute difference between the two images
            difference = cv2.absdiff(gray2, gray1)
            # Apply thresholding to highlight the differences
            _, thresholded = cv2.threshold(difference, 110, 255,
↪ cv2.THRESH_BINARY)
            # Find contours of the differences
            contours, _ = cv2.findContours(thresholded, cv2.
↪ RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            # Draw rectangles around the differing regions
            for contour in contours:
                x, y, w, h = cv2.boundingRect(contour)
                source_img = cv2.cvtColor(np.array(source_img), cv2
↪ .COLOR_BGR2RGB)
                cv2.rectangle(source_img, (x, y), (x + w, y + h),
↪ (255, 0, 0), 5)


            st.image(source_img, caption='Tested PCB')
```

```
            else:
                st.sidebar.write("Please upload both a reference image
    ↪ and a source image.")

def Component_Defects_live():

    st.subheader('Live')
    #st.title('PCB Defects Detection')

    # URL to fetch continuous images
    url = 'http://192.168.182.214/cam-hi.jpg'

    # Display the live stream
    image_placeholder = st.empty()  # Placeholder to update the image

    # Flag to track if the button is clicked
    button_clicked = False
    Rbutton_clicked = False


    # Create the button to process image

    Reference_button = st.button('Set Reference Image')
    process_button = st.button('Process Image')

    col1, col2 = st.columns(2)


    while True:

        with col1:
            if Reference_button:
                Rbutton_clicked = True
                Reference_button = False

            response = requests.get(url)
            Reference_image = Image.open(BytesIO(response.content))
            reference_img = cv2.cvtColor(np.array(Reference_image), cv2
    ↪ .COLOR_RGB2BGR)

            if Rbutton_clicked:
                st.image(Reference_image,caption='Reference Image')

                Rbutton_clicked = False
                time.sleep(2)

        with col2:
```

```
        if process_button:
            button_clicked = True
            process_button = False

        response = requests.get(url)
        image = Image.open(BytesIO(response.content))

        if button_clicked:
            #st.image(image)

            button_clicked = False
            time.sleep(2)

            source_img = cv2.cvtColor(np.array(image), cv2.
↪ COLOR_RGB2BGR)

            gray1 = cv2.cvtColor(reference_img, cv2.COLOR_BGR2GRAY)
            gray2 = cv2.cvtColor(source_img, cv2.COLOR_BGR2GRAY)
                # Compute absolute difference between the two
↪ images
            difference = cv2.absdiff(gray1, gray2)
                # Apply thresholding to highlight the differences
            _, thresholded = cv2.threshold(difference, 10, 255, cv2
↪ .THRESH_BINARY)
                # Find contours of the differences
            contours, _ = cv2.findContours(thresholded, cv2.
↪ RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
                # Draw rectangles around the differing regions
            for contour in contours:
                x, y, w, h = cv2.boundingRect(contour)
                cv2.rectangle(reference_img, (x, y), (x + w, y + h)
↪ , (255, 0, 0), 5)

            st.image(reference_img, caption='Component Defects')
            st.write("dannan source")



            time.sleep(5)









        else:
        # Display the image without processing
```

```
                image_placeholder.image(image, channels="RGB")

        # Add a small delay to control the frame rate
        time.sleep(0.001)  # 0.1 second delay



if __name__ == '__main__':
                main()
```

10 References

1. Hubei University Of Technology, PCB Dataset , November 2023
URL: https://universe.roboflow.com/hubei-university-of-technology-rmbpi/pcb-ecjg (visited on 2024-05-31)

2. Eric Nam, Enable OV5640's autofocus function on ESP32 AI-THINKER Board, 2021
https://github.com/0015/ESP32-OV5640-AF/blob/main/LICENSE

3. Free Software Foundation,Ultralytics YOLOv8, 2007
https://github.com/ultralytics/ultralytics/blob/main/LICENSE

4. ESP32-WROOM-32 (ESP-WROOM-32) Datasheet, Version 2.4
https://www.mouser.com/datasheet/2/891/esp-wroom-32_datasheet_en-1223836

5. OV2640 Color CMOS UXGA (2.0 MegaPixel) CAMERACHIP, Preliminary Datasheet
https://ndatasheet.com/view/548838/OmniVisionTechnologies/OV2640/1