PCB (Printed Circuit Board) Defect Detection

Contents

Time required: 120 minutes

What are PCB Defects?

PCB (Printed Circuit Boards) defect detection is a crucial process in ensuring the quality and reliability of printed circuit boards (PCBs). It involves identifying and classifying various types of defects that can occur during the manufacturing process.

Types of Defects

Common defects in PCBs include:

- **Missing Hole**: Absence of a required hole.
- **Mouse Bite**: Small, irregular cuts or nibbles on the edges.
- **Open Circuit**: Breaks in the conductive path.
- **Short Circuit**: Unintended connections between conductive paths.
- **Spur**: Unwanted protrusions of copper.
- **Spurious Copper**: Random copper deposits

Detection Methods

1. **Visual Inspection**: Manual inspection using microscopes or magnifying glasses.
2. **Automated Optical Inspection (AOI)**: Uses cameras and software to detect defects.
3. **X-ray Inspection**: Identifies internal defects not visible on the surface.
4. **Electrical Testing**: Checks for continuity and proper electrical connections

PCB Image Files

We are going to use Optical Inspection with MATLAB.

1. Download the following images[1].
    a. pcb_template.jpg is the template image.
    b. pcb_defect.jpg is the defect image.
2. Upload these files to MATLAB.

[1] https://github.com/amansoni000/PCB-Defect-Detection-Using-Image-Processing

MATLAB PCB Defect Detection Program

NOTE: Test each code set to make sure it works. After that, you can comment out the imshow() command.

```
% Use image processing to detect PCB defects
clc % Clear the command window
```

Image Loading

- **Function**: imread
- **Purpose**: To load the images of the PCB boards into MATLAB.
- **Explanation**: The imread function reads an image file and stores it as a matrix in MATLAB. Each element of the matrix represents a pixel in the image.

```
%% Load the two PCB images
% Load the template PCB image (defect-free)
imageTemplate = imread('pcb_template.jpg');
% Load the test PCB image (potentially with defects)
imageDefect = imread('pcb_defect.jpg');
imshow(imageDefect);
```

Grayscale Conversion

- **Function**: rgb2gray
- **Purpose**: To convert RGB images to grayscale.
- **Explanation**: The rgb2gray function converts a color image (RGB) to a grayscale image. Grayscale images are easier to process and analyze for defects.

```
%% Convert images to grayscale
grayImageTemplate = rgb2gray(imageTemplate);
grayImageDefect = rgb2gray(imageDefect);
imshow(grayImageDefect);
```

Gaussian Blur

- **Function**: imgaussfilt
- **Purpose**: To apply Gaussian blur to reduce noise.

- **Explanation**: The imgaussfilt function applies a Gaussian filter to smooth the image and reduce noise. This makes defect detection more accurate by minimizing irrelevant variations.

```
%% Apply Gaussian blur to reduce noise
blurredImageTemplate = imgaussfilt(grayImageTemplate, 2);
blurredImageDefect = imgaussfilt(grayImageDefect, 2);
imshow(blurredImageDefect);
```

Difference Calculation

- **Function**: abs
- **Purpose**: To calculate the absolute difference between the two images.
- **Explanation**: The abs function computes the absolute value of the difference between corresponding pixels in the two images. This highlights areas where the images differ, which may indicate defects.
  Each image is a matrix. If we subtract one from the other, we end up with the parts that are different in a new matrix.

```
%% Calculate the absolute difference between the blurred images
diffImage = abs(blurredImageTemplate - blurredImageDefect);
imshow(diffImage);
```

Thresholding

- **Purpose**: To create a binary image that highlights defects.
- **Explanation**: Thresholding converts the difference image into a binary image, where pixels with values above a certain threshold are set to 1 (white), and others are set to 0 (black). This helps in isolating the defects. 50 is a good starting value.

```
%% Threshold the difference image to highlight defects
% Set a threshold value; lower values detect more defects
threshold = 50;
% Create binary image where pixels above the threshold are set to 1 (white)
binaryDiffImage = diffImage > threshold;
imshow(binaryDiffImage);
```

Labeling

- **Function**: logical
- **Purpose**: To convert the binary image into a logical array for further processing.

- **Explanation**: The logical function converts the binary image into a logical array, where each element is either true (1) or false (0). This is useful for identifying and processing connected components (defects) in the image.

```
% Convert the binary image to a logical array
labeledImage = logical(binaryDiffImage);
```

Visualization

- **Functions**: figure, subplot, imshow, title
- **Purpose**: To display the original images and the binary difference image.
- **Explanation**: These functions create a figure window and display the images in subplots. imshow displays an image, and title adds a title to each subplot.

```
%% Display the original images and the difference image
figure; % Create a new figure window

subplot(2, 2, 1); % Create a subplot for the first image
imshow(imageTemplate); % Display the template PCB image
title('PCB Image Template');

subplot(2, 2, 2); % Create a subplot for the second image
imshow(imageDefect); % Display the test PCB image
title('PCB Image Defect');

subplot(2, 2, 3); % Create a subplot for the difference image
imshow(binaryDiffImage); % Display the binary difference image
title('Detected Defects on Binary Diff Image');

subplot(2,2,4);
imshow(imageDefect); % Display the test PCB image
title('Detected Defects on PCB Image');
```

Bounding Boxes

- **Functions**: regionprops, rectangle
- **Purpose**: To draw bounding boxes around detected defects.
- **Explanation**: The regionprops function measures properties of labeled regions in the image, such as the bounding box.

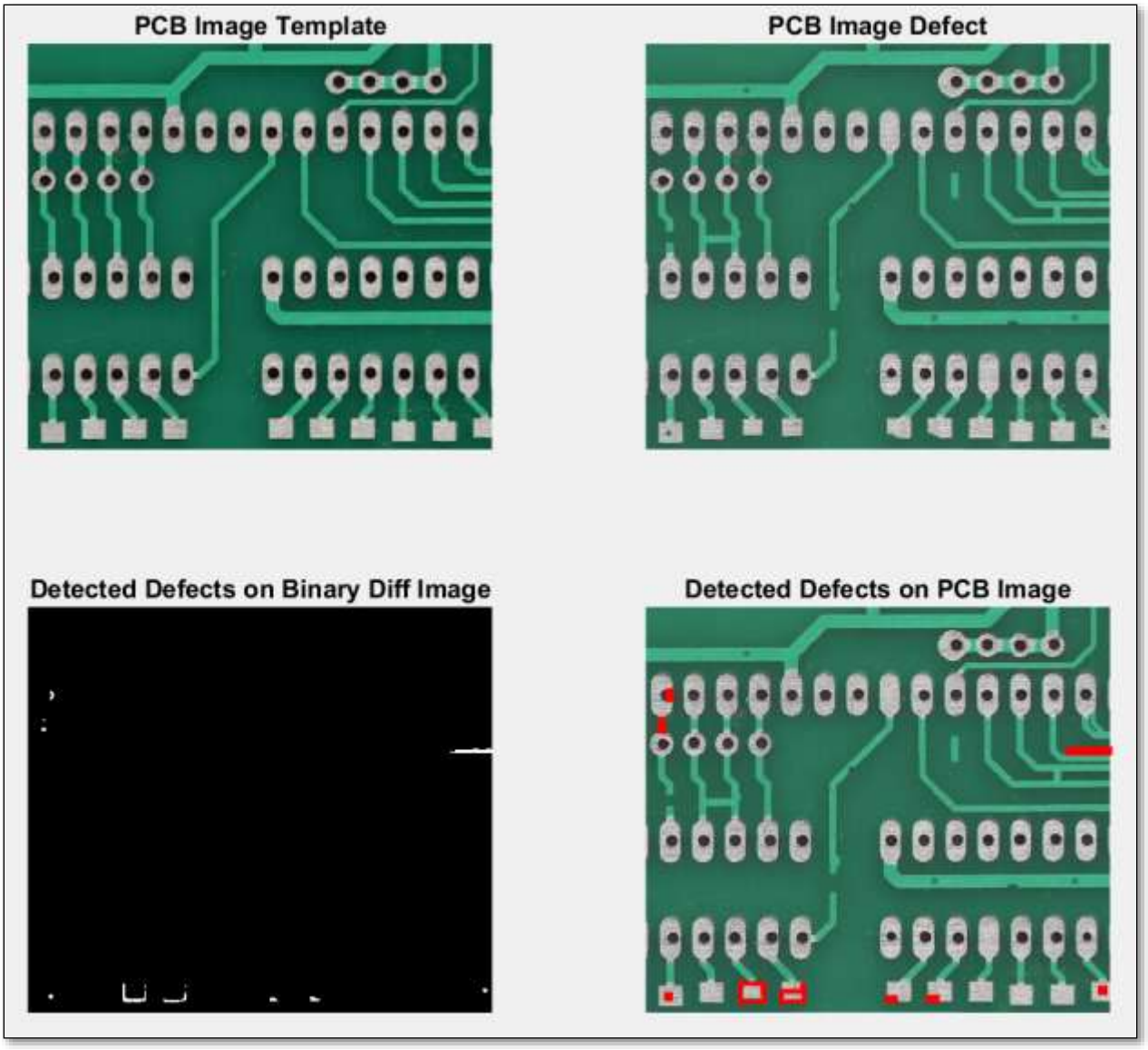
- The rectangle function draws rectangles around these regions to highlight the defects.
  rectangle('Position', position, 'EdgeColor', color, 'LineWidth', width)

```
%% Overlay bounding boxes on the original images
% Get the bounding boxes of the labeled regions (defects)
stats = regionprops(labeledImage, 'BoundingBox');

% Loop through each bounding box
for i = 1:length(stats)
    % Get the current bounding box
    bbox = stats(i).BoundingBox;

    % Draw a red rectangle around the detected defect
    rectangle('Position', bbox, 'EdgeColor', 'r', 'LineWidth', 2);
end
```

Example run:



Assignment 1: Adjust Threshold to Show all Defects

In this MATLAB program, the threshold is 50. This is a good starting point. This shows most defects, but not all the defects.

1. What threshold value shows all defects, without showing false positives?
2. Add a screenshot of the image that you think shows defects without showing false positives.
3. Attach to assignment.

Python PCB Defect Detection Program

Python has an image manipulation library called OpenCV. With OpenCV and MatPlotLib, we can create a PCB defect detection program in Google Colab. As you go through this tutorial, notice how the Python code resembles the MATLAB code.

1. Create a Google Colab Notebook: **PCBDefectDetectionProgram.ipynb**

Step 1: Import Libraries

- **Libraries**: cv2, matplotlib.pyplot, skimage.measure
- **Purpose**: To import the necessary libraries for image processing and visualization.
- **Explanation**: We import the OpenCV (cv2) library for image processing, Matplotlib for plotting, and Scikit-Image for measuring properties of labeled regions.

```
import cv2
import matplotlib.pyplot as plt
from skimage import measure
import requests
import numpy as np
```

Step 2: Load the PCB Images

Loading an image into Google Colab is a bit complicated. Here is the code. Copy and paste into a Google Colab code cell.

```
def download_image(url):
    # Send a GET request to the specified URL
    response = requests.get(url)

    # Check if the request was successful (status code 200)
    if response.status_code == 200:
        # Convert the response content (image data) into a NumPy array
        image_array = np.frombuffer(response.content, np.uint8)

        # Decode the image array into an image using OpenCV
        image = cv2.imdecode(image_array, cv2.IMREAD_COLOR)

        # Return the decoded image
        return image
    else:
        # Print an error message if the image could not be downloaded
        print("Error: Unable to download image.")

        # Return None to indicate failure
        return None

# Load the template PCB image (defect-free)
image_template = download_image('https://github.com/itinstructor/WNCCComputerScience/raw/main/MATLAB/Assignments/pcb_template.jpg')
# Load the test PCB image (potentially with defects)
image_defect = download_image('https://github.com/itinstructor/WNCCComputerScience/raw/main/MATLAB/Assignments/pcb_defect.jpg')

# Check if images are loaded correctly
if image_template is None or image_defect is None:
    print("Error: One or both images could not be loaded. Check the file paths.")
else:
    print("Images loaded successfully.")
```

Step 3: Convert Images to Grayscale

- **Function**: cv2.cvtColor
- **Purpose**: To convert RGB images to grayscale.
- **Explanation**: The cv2.cvtColor function converts a color image (RGB) to a grayscale image. Grayscale images are easier to process and analyze for defects.

```
# Convert images to grayscale
gray_image_template = cv2.cvtColor(image_template, cv2.COLOR_BGR2GRAY)
gray_image_defect = cv2.cvtColor(image_defect, cv2.COLOR_BGR2GRAY)
```

Step 4: Apply Gaussian Blur

- **Function**: cv2.GaussianBlur
- **Purpose**: To apply Gaussian blur to reduce noise.
- **Explanation**: The cv2.GaussianBlur function applies a Gaussian filter to smooth the image and reduce noise. This makes defect detection more accurate by minimizing irrelevant variations.

```
# Apply Gaussian blur to reduce noise
blurred_image_template = cv2.GaussianBlur(gray_image_template, (5, 5), 2)
blurred_image_defect = cv2.GaussianBlur(gray_image_defect, (5, 5), 2)
```

Step 5: Calculate the Absolute Difference

- **Function**: cv2.absdiff
- **Purpose**: To compute the absolute difference between the images.
- **Explanation**: The cv2.absdiff function calculates the absolute value of the difference between corresponding pixels in the two images. This highlights areas where the images differ, which may indicate defects.

```
# Calculate the absolute difference between the blurred images
diff_image = cv2.absdiff(blurred_image_template, blurred_image_defect)
```

Step 6: Threshold the Difference Image

- **Function**: cv2.threshold
- **Purpose**: To create a binary image that highlights defects.
- **Explanation**: Thresholding converts the difference image into a binary image, where pixels with values above a certain threshold are set to 1 (white), and others are set to 0 (black). This helps in isolating the defects.

```
# Threshold the difference image to highlight defects
threshold = 50
_, binary_diff_image = cv2.threshold(diff_image, threshold, 255, cv2.THRESH_BINARY)
```

Step 7: Convert to Logical Array

- **Function**: astype

- **Purpose**: To convert the binary image into a logical array for further processing.
- **Explanation**: The astype function converts the binary image into a logical array, where each element is either true (1) or false (0). This is useful for identifying and processing connected components (defects) in the image.

```
# Convert the binary image to a logical array for further processing
labeled_image = binary_diff_image.astype(bool)
```

Step 8: Display the Images

- **Libraries**: matplotlib.pyplot
- **Functions**: plt.figure, plt.subplot, plt.imshow, plt.title, plt.show
- **Purpose**: To display the original images and the binary difference image.
- **Explanation**: These functions create a figure window and display the images in subplots. plt.imshow displays an image, and plt.title adds a title to each subplot.

```
# Display the original images and the difference image
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(image_template, cv2.COLOR_BGR2RGB))
plt.title('PCB Image Template')

plt.subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(image_defect, cv2.COLOR_BGR2RGB))
plt.title('PCB Image Defect')

plt.subplot(1, 3, 3)
plt.imshow(binary_diff_image, cmap='gray')
plt.title('Detected Defects')

plt.show()
```

Step 9: Overlay Bounding Boxes

- **Libraries**: skimage.measure
- **Functions**: measure.label, measure.regionprops, cv2.rectangle
- **Purpose**: To draw bounding boxes around detected defects.

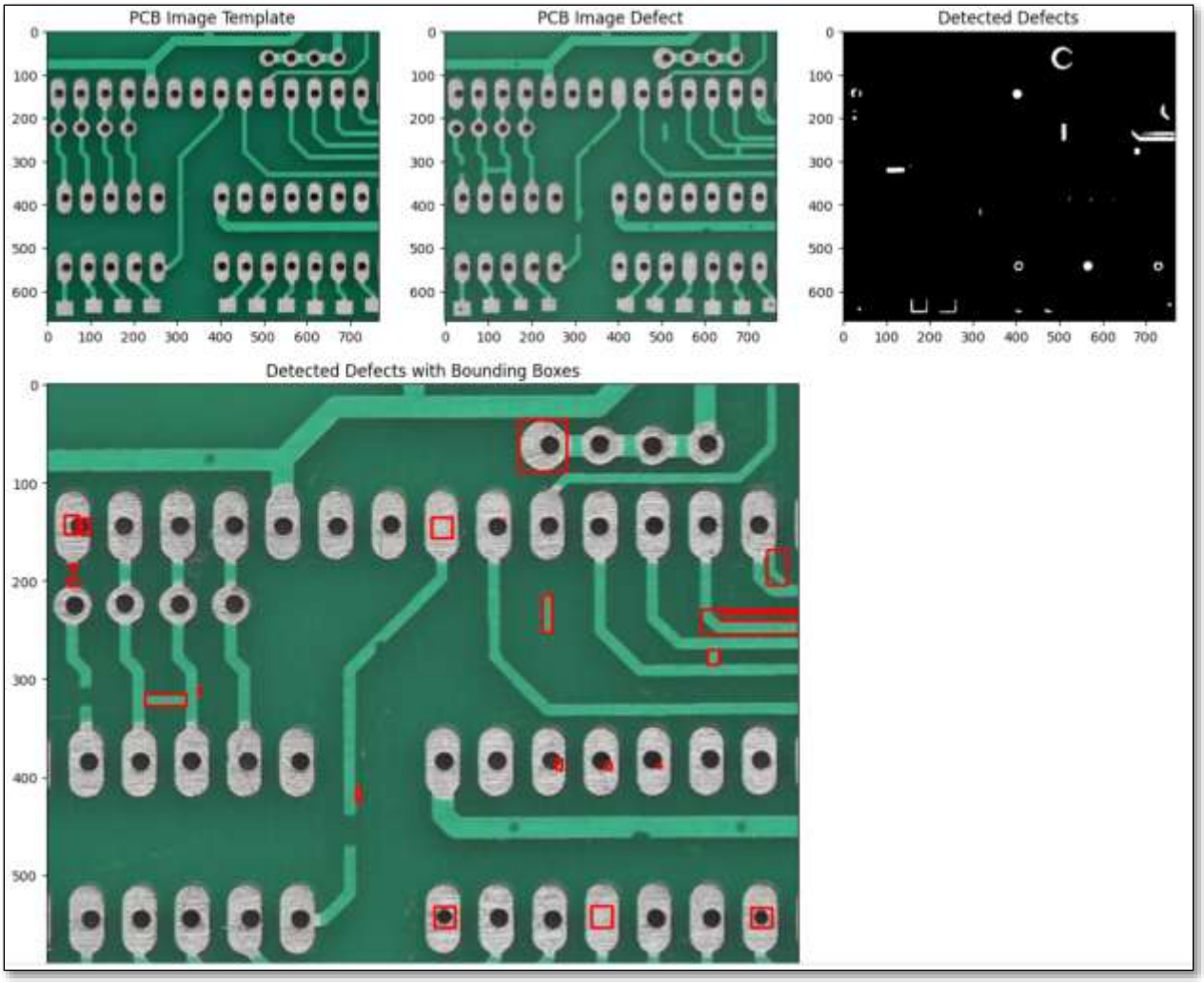
- **Explanation**: The measure.label function labels connected components in the image. The measure.regionprops function measures properties of labeled regions, such as the bounding box. The cv2.rectangle function draws rectangles around these regions to highlight the defects.

```
# Overlay bounding boxes on the original images
# Get the bounding boxes of the labeled regions (defects)
labels = measure.label(labeled_image)
props = measure.regionprops(labels)

# Draw bounding boxes on the defect image
for prop in props:
    bbox = prop.bbox
    cv2.rectangle(image_defect, (bbox[1], bbox[0]), (bbox[3], bbox[2]), (0, 0, 255), 2)

# Display the defect image with bounding boxes
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(image_defect, cv2.COLOR_BGR2RGB))
plt.title('Detected Defects with Bounding Boxes')
plt.show()
```

Example run:



Assignment 1: Adjust Threshold to Show all Defects

In this Python program, the threshold is 50. This is a good starting point. Notice that this shows more defects than MATLAB at this threshold. This shows most defects, but not all the defects.

1. What threshold value shows all defects, without showing false positives?
2. Add a screenshot of the image that you think shows defects without showing false positives.
3. Attach to assignment.

Assignment Submission

1. Submit properly named and commented script file.

2. Attach a Word document with a screenshot showing the successful execution of the script.
    a. Show a screenshot with your adjusted threshold value.
    b. Include the value.
3. Attach all to the assignment in Blackboard.