Barcode Detection and Decoding using Image Processing

Bhavin Gurnani - 202418018
Shaili Parekh - 202418049

Master's in Data Science
Advanced Image Processing Project Submission

Abstract

This project explores barcode detection and decoding using classical image processing techniques. We implement a pipeline using OpenCV and the ZBar library to identify, extract, and decode 1D barcodes from images. The system is designed to be simple and efficient under ideal conditions, with potential for extension to handle more complex cases such as noisy or rotated images.

1 Introduction

Barcodes are prevalent across industries for tasks such as inventory tracking, product identification, and data encoding. Despite their simplicity, extracting and decoding barcodes from raw images poses challenges due to variations in lighting, orientation, noise, and image quality.

This project addresses the problem of reliably detecting and decoding 1D barcodes from images using a combination of classical image processing techniques and barcode decoding libraries.

2 Objective

The main goal of this project is to develop a pipeline that can:

- Identify barcode regions in an input image.
- Extract and preprocess the barcode area.
- Decode the barcode into a readable format.

3 Tools and Libraries Used

- Python 3
- OpenCV
- NumPy
- pyzbar (ZBar Python wrapper)

4 Methodology

The barcode detection and decoding pipeline follows these steps:

1. Read the input image using OpenCV.
2. Convert the image to grayscale.
3. Apply gradient filters to emphasize the barcode structure.
4. Use morphological operations to reduce noise and enhance the barcode area.

5. Detect contours and extract potential barcode regions.
6. Use the ZBar library (via pyzbar) to decode the barcode.
7. Display the image with the decoded result overlaid.

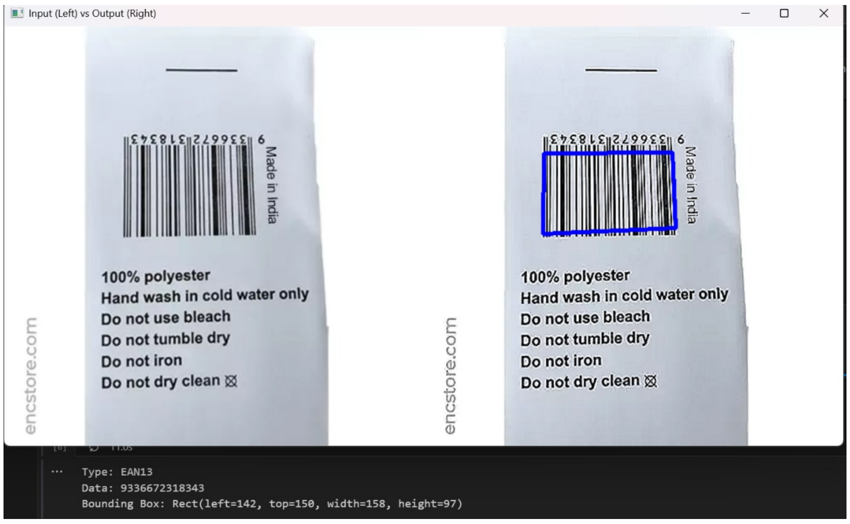
5 Pseudocode

```
Function: Detect_And_Decode_Barcode(Image)
    Read the input image
    Convert image to grayscale
    Compute gradient using Sobel or Scharr operator
    Subtract vertical gradient from horizontal gradient
    Apply Gaussian blur to smooth image
    Apply binary thresholding
    Perform morphological operations (closing, erosion, dilation)
    Find contours in the image
    For each contour:
        Compute bounding box
        Extract Region of Interest (ROI)
        Decode barcode using pyzbar.decode()
        If decoded: draw bounding box and overlay text
    Display final image with barcode annotation
```

6 Output

The output of this project comprises both visual and textual results.

- **Visual Output**: The processed image displays the original input with the barcode region highlighted, typically using a bounding box. Additionally, the decoded barcode information is overlaid on the image for easy identification.
- **Textual Output**: The decoded barcode data are presented in a readable format which can be displayed in the console or saved to a file for further use.



```
Type: EAN13
Data: 9336672318343
Bounding Box: Rect(left=142, top=150, width=158, height=97)
```

Figure 1: Barcode Detection: Input (Left) vs Output (Right)

7 Conclusion

This project demonstrates a successful implementation of barcode detection and decoding using classical image processing techniques. The combination of OpenCV for image preprocessing and ZBar for decoding is effective under standard conditions.

Future improvements could include:

- Handling rotated and skewed barcodes.
- Improve robustness against noise and poor lighting.
- Extending to support 2D barcodes like QR codes.

References

- OpenCV Documentation: `https://docs.opencv.org/`
- Pyzbar Documentation: `https://github.com/NaturalHistoryMuseum/pyzbar`