Name: Sahan Rudrigo
Date: 21st April 2025

Enhancing Transparent Spoon Detection

-A Comparative Analysis of Detection Before and After Image Preprocessing-

Introduction

This report evaluates the transparent spoon detection system developed using OpenCV and Python. The system aims to detect transparent spoons in images using template matching, with a focus on the preprocessing steps that enhance detection accuracy. Transparent spoons are challenging to detect due to their low contrast and tendency to blend into the background. This report compares the system's performance before and after preprocessing, specifically for transparent spoons, and provides visual evidence.

Overview of the Detection System

I. Loading the Template and Input Image

II. Preprocessing

III. Template Matching

IV. GUI Display

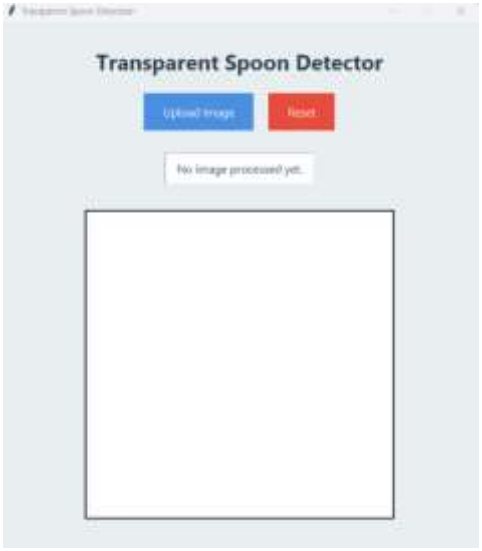

Figure 1-simple GUI

Before Preprocessing

Without preprocessing, the detection system struggles to identify transparent spoons due to the following challenges:

- **Low Contrast**: Transparent spoons have minimal intensity differences from the background, making them nearly invisible in grayscale.
- **Background Noise**: Textured backgrounds introduce noise that interferes with template matching.
- **Feature Absence**: The lack of distinct edges or intensity patterns in the raw image makes it difficult for template matching to find a good correlation

Detection Attempt Without Preprocessing

If we skip the preprocessing step and directly apply template matching on the raw images (after converting to grayscale for consistency), the results are poor:

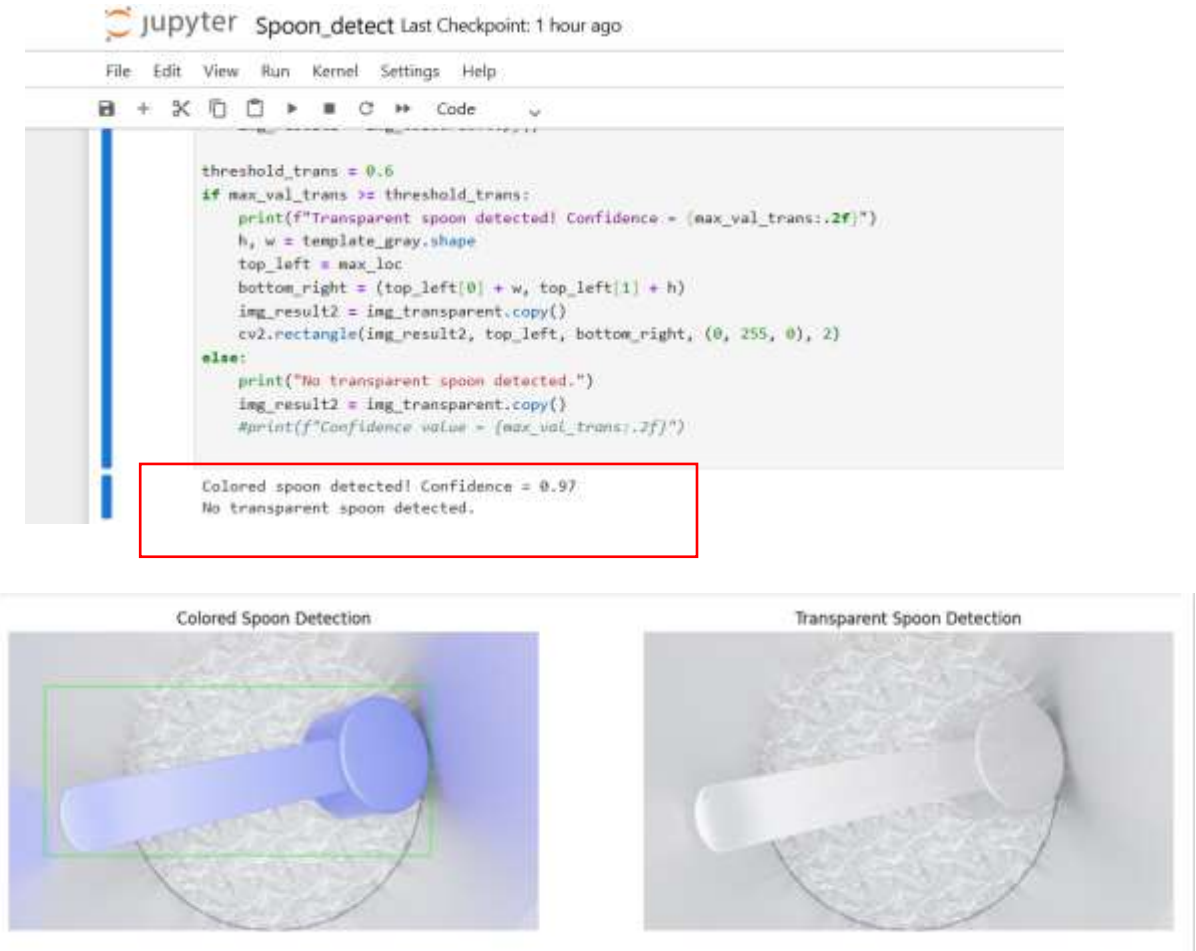

Figure 2-detection without preprocess

Preprocessing Steps

To address the challenges, a preprocessing pipeline was developed to enhance the features of both the input image and the template. The updated preprocess_image function includes the following steps:

1. **Convert to Grayscale**: The image is converted from color (BGR) to grayscale using cv2.cvtColor(image, cv2.COLOR_BGR2GRAY). This simplifies the image to a single intensity channel, making it easier to process while focusing on brightness differences and increase processing speed

Figure 3-covert to gray scale

2. **Noise Reduction (Bilateral Filter + Gaussian Blur)**: A bilateral filter (cv2.bilateralFilter) with a diameter of 9 and sigma values of 100 is applied to reduce noise while preserving edges. This is followed by a Gaussian blur (cv2.GaussianBlur) with a 5x5 kernel and a sigma of 10 to further smooth the image, reducing background texture noise like crumpled foil patterns.

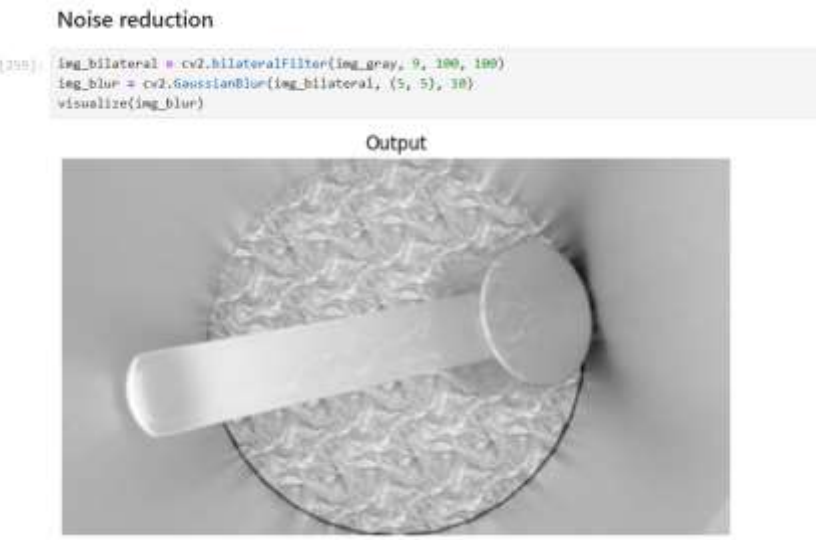

Figure 4-Noise Reduction

3. **Enhance Contrast (CLAHE)**: Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied using cv2.createCLAHE with a clipLimit of 4.0 and a tileGridSize of 5x5. This enhances local contrast, making the transparent spoon's faint outline more distinguishable against the background.

Figure 5- Enhance Contrast

4. **Edge Detection (Canny)**: The Canny edge detector (cv2.Canny) is used with low thresholds of 78 and 80 to detect faint edges of the transparent spoon. These lowered thresholds increase sensitivity to subtle edges, which are critical for transparent objects.

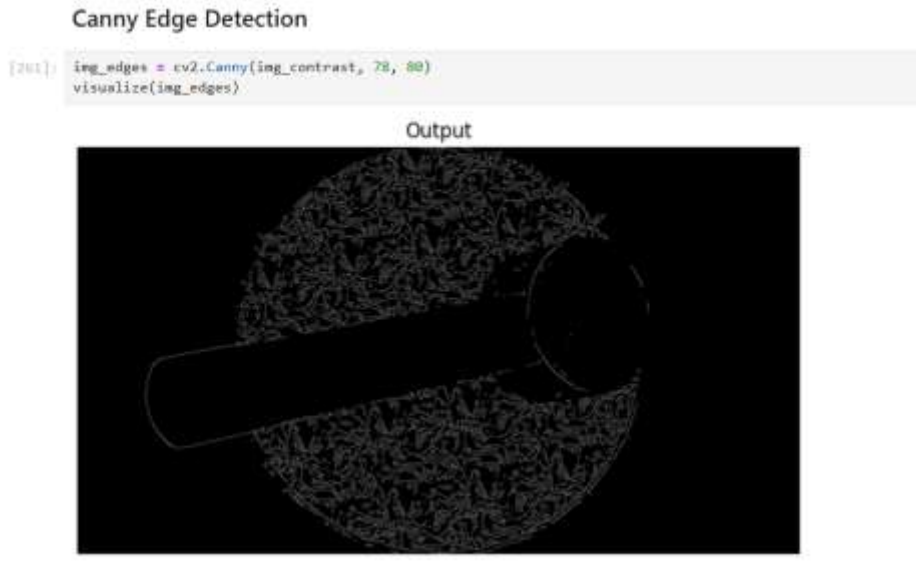

Figure 6-Edge detector

5. **Morphological Closing**: A morphological closing operation (cv2.morphologyEx) with a 6x6 kernel and 3 iterations is applied to connect disjointed edges of the spoon. This strengthens the spoon's outline, making it more suitable for template matching.

Morphological Closing

```
[264]: kernel = np.ones((6, 6), np.uint8)
       img_closed = cv2.morphologyEx(img_edges, cv2.MORPH_CLOSE, kernel, iterations=3)
       visualize(img_closed)
```

Output

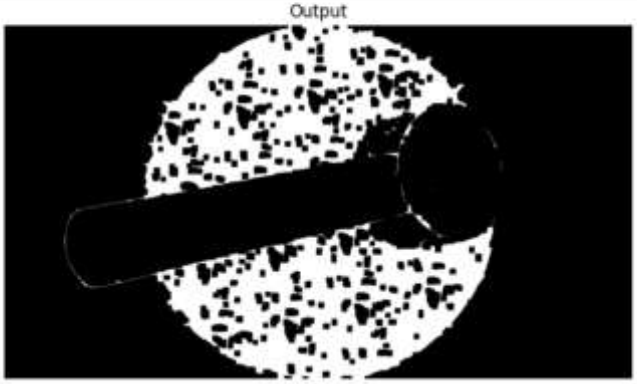

Figure 7-Morphological Closing

Function for preprocess

```
[265]: def preprocess_image(image):
           # Step 1: Convert to grayscale
           img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

           # Step 2: Noise reduction (bilateral filter + Gaussian blur)
           img_bilateral = cv2.bilateralFilter(img_gray, 4, 100, 100)
           img_blur = cv2.GaussianBlur(img_bilateral, (5, 5), 10)

           # Step 3: Enhance contrast (stronger CLAHE)
           clahe = cv2.createCLAHE(clipLimit=4.0, tileGridSize=(5, 5))
           img_contrast = clahe.apply(img_blur)

           # Step 4: Edge detection (lower thresholds)
           img_edges = cv2.Canny(img_contrast, 78 , 80)  # Lowered for faint edges

           # Step 5: Morphological closing (stronger)
           kernel = np.ones((6, 6), np.uint8)  # Larger kernel
           img_closed = cv2.morphologyEx(img_edges, cv2.MORPH_CLOSE, kernel, iterations=3)

           return img_closed
```

Figure 8- Function for preprocessing

After Preprocessing

With preprocessing, the detection system performs significantly better for transparent spoons:

Figure 9-Performance after preprocessing

Challenges with Transparent Spoons

- **Lack of Color/Intensity Differences**: Transparent spoons blend into the background. **Solution**: CLAHE enhances contrast, making the spoon's outline more visible.

- **Faint Edges on Noisy Backgrounds**: Edges are weak, especially on reflective surfaces like crumpled foil. **Solution**: Canny edge detection with low thresholds captures faint edges, and bilateral filter/Gaussian blur reduce noise.

- **Template Mismatch**: The colored spoon template differs from the transparent spoon. **Solution**: Preprocessing both the template and input image (starting with grayscale conversion) aligns their features, and morphological closing strengthens the spoon's outline, improving matching. (Figure 10)

Figure 10

Demo Video:

https://drive.google.com/file/d/1V-PUI1hIRe7ymDiMlgmmreakRhdSto1t/view?usp=sharing

References

https://youtu.be/ADV-AjAXHdc?si=U8-s6saI4NhP0D6h