FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELECTRONIK DAN KOMPUTER (FTKEK)

SEM 1 2024/2025

BERR 4723

DIGITAL IMAGE PROCESSING

4 BENR S3

PROJECT TITLE:

BACKGROUND REMOVAL SYSTEM USING DIGITAL IMAGE PROCESSING

LECTURER:

PROFESOR MADYA DR NURULFAJAR BIN ABD MANAP

STUDENT NAME	MATRIC NUMBER
AMIR AFZAL BIN CHE RASHID	B022120016
KANG CHIAW NA	B022120004
WAN SYAHIRAH AKMAL BIN WAN SAUPI	B022120030
SITI SHAFAWATIH BINTI SALEH	B022120024

1.0 INTRODUCTION

This Background Removal System Using a Digital Image Processing project is designed to create an accurate and efficient solution for isolating and removing backgrounds from digital images. This system is designed to enhance the quality of images by applying a series of digital image processing techniques where each contributing to the overall effectiveness of the background removal process. This comprehensive approach of techniques makes the system versatile and efficient which is suitable for applications in photography, graphic design and computer vision.

The project begins with **geometric transformations** such as resizing and scaling. **Image enhancement** methods like histogram equalization to improve contrast and visibility. **Image restoration** techniques including noise removal will help eliminate unwanted artifacts to ensure clarity. The project also employs **image segmentation** to distinguish and separate elements from the background. **Colour space transformations** are utilized to manipulate colour information for more accurate background removal. Finally, **compression** techniques are integrated to reduce the file size of processed images without reducing the quality. Challenges encountered during implementation such as the difficulty of incorporating transparency due to limitations in common colour models, ensuring stability in the multi-step image processing and maintaining compatibility across various image formats.

2.0 METHODOLOGY

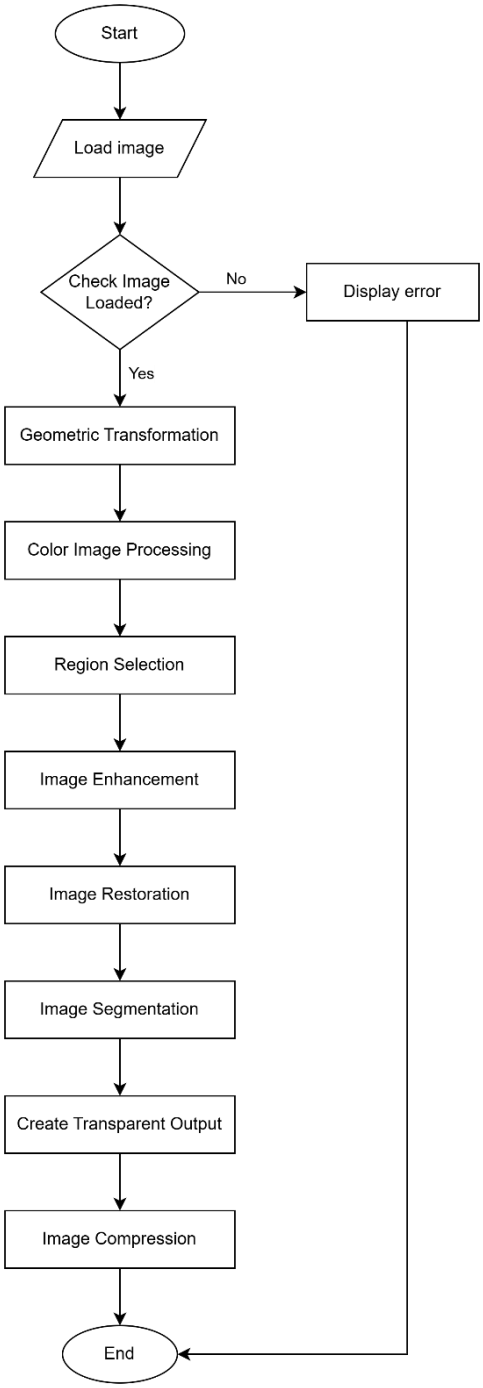

Figure 1: Flowchart of the system

i. Input image

The system will first load the input image to check if the image is loaded successfully. An error message will show if the image fails to load.

```
img = cv2.imread(r"C:\Users\ASUS\OneDrive\Sem6\Digital Image Processing\assignment\green_apple.jpg")

if img is None:
    print("Error: Image not found or unable to load.")
```

Figure 2: Code to check the loaded image

ii. Geometric Transformation

The Python code as shown in Figure 3 is used to resize and change the dimensions of the input image to a new width and height.

```
else:
    # Resize the image (optional step)
    new_width = 600  # Set desired width
    new_height = 400  # Set desired height
    img = cv2.resize(img, (new_width, new_height))
```

Figure 3: Code to resize the image

iii. Colour Image Processing

The resized image is converted to BGRA format by adding an alpha channel to the image to handle transparency.

```
# Convert to BGRA format (with an alpha channel)
b_channel, g_channel, r_channel = cv2.split(img)
alpha_channel = np.ones(b_channel.shape, dtype=b_channel.dtype) * 255  # Fully opaque
img_bgra = cv2.merge((b_channel, g_channel, r_channel, alpha_channel))  # Merge BGR channels with alpha
```

Figure 4: Color Space Transformation

iv. Region Selection

The user is allowed to draw a bounding box using box events to select the region of interest (ROI).

```
def draw_rectangle(event, x, y, flags, param):
    global ix, iy, drawing, img, bx, by
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            img_copy = img.copy()
            cv2.rectangle(img_copy, (ix, iy), (x, y), (0, 255, 0), 2)
            cv2.imshow("Image", img_copy)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        bx, by = x, y  # Save the final rectangle's bottom-right corner
        cv2.rectangle(img, (ix, iy), (bx, by), (0, 255, 0), 2)
        cv2.imshow("Image", img)
```

Figure 5: Region Selection

v. Image Enhancement

Then, the image with the selected region is converted to grayscale to focus on intensity values. Besides, grayscale is essential for tasks like histogram equalization and thresholding as they rely on intensity values.

```
# Step 2: Convert only the selected region to grayscale
gray_region = cv2.cvtColor(selected_region, cv2.COLOR_BGR2GRAY)
cv2.imshow("Grayscale Region", gray_region)  # Show grayscale image
cv2.waitKey(0)
```

Figure 6: RGB to Grayscale conversion

The histogram equalization is applied to improve visibility by redistributing the intensity values.

```
# Step 3.1: Apply histogram equalization to make the object darker (enhance contrast)
equalized_region = cv2.equalizeHist(blurred_region)
cv2.imshow("Equalized Region", equalized_region)  # Show equalized image
cv2.waitKey(0)
```

Figure 7: Histogram Equalization

vi. Image Restoration

The Gaussian blur is used to reduce the noise and then smoothen the grayscale image to prepare for segmentation.

```
# Step 3: Apply Gaussian blur to reduce noise
blurred_region = cv2.GaussianBlur(gray_region, (5, 5), 0)
```

Figure 8: Noise Removal

vii. Image Segmentation

Then, the image is segmented into foreground and background using a binary threshold and reverses the black and white regions to focus on the desired area.

```
# Step 4: Perform thresholding to create a binary image (only white and black)
_, binary_region = cv2.threshold(equalized_region, 190, 210, cv2.THRESH_BINARY)
cv2.imshow("Binary Region", binary_region)  # Show binary image
cv2.waitKey(0)

# Step 5: Invert the binary image (white to black and black to white)
inverted_region = cv2.bitwise_not(binary_region)
cv2.imshow("Inverted Region", inverted_region)  # Show inverted image
cv2.waitKey(0)
```

Figure 9: Image Segmentation

viii. Create Transparent Output

Next, use the binary mask to make the segmented region transparent and combine the ROI with the alpha mask to create a transparent BGRA image.

```
# Step 6: Create an alpha mask where white is 0 (transparent) and black is 255 (opaque)
alpha_mask = inverted_region  # This mask now ensures the object is removed (transparent)
alpha_mask = alpha_mask.astype(np.uint8)

# Step 7: Apply the alpha mask to the color channels (B, G, R) of the selected region
for c in range(3):  # Iterate over color channels (0: Blue, 1: Green, 2: Red)
    selected_region[:, :, c] = selected_region[:, :, c] * (alpha_mask / 255.0)

# Step 8: Create a new image (BGRA) for saving the selected region
selected_region_bgra = cv2.merge((selected_region[:, :, 0], selected_region[:, :, 1], selected_region[:, :, 2], alpha_mask))
```

Figure 10: Create Transparent Output

ix. Image Compression

Lastly, save the processed image with PNG compression to reduce file size while preserving quality.

```
# Step 9: Compress and save the selected region with transparency as a PNG file
png_compression_params = [cv2.IMWRITE_PNG_COMPRESSION, 5]  # Level 5 compression (out of 0-9)
cv2.imwrite("Cropped_PNG_Image.png", selected_region_bgra, png_compression_params)

# Display the saved PNG image with transparency
cv2.imshow("Saved PNG Image", selected_region_bgra)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figure 11: Image Compression

3.0 RESULTS AND ANALYSIS

The results obtained in this project are captured after each process of digital image processing technique to analyse the method application and its usage to remove the background. Each process is crucial and need to follow the procedure to receive the desire result.

i. Geometric Transformation

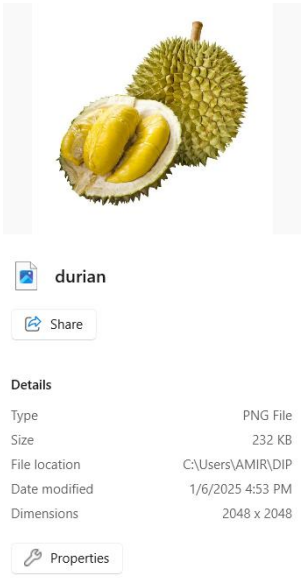

Figure 12: The original image size before resizing.

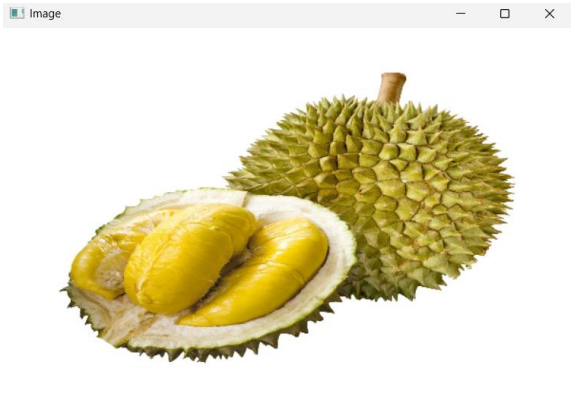

Figure 13: The image after resizing to 600x400

The uploaded images were resizing to a uniform dimension of 600x400 pixels. This technique is necessary in preventing errors when handing with images with different dimension. The technique also helps with consistent user interface (UI) experience across the system.

ii. Image Enhancement (Grayscale)

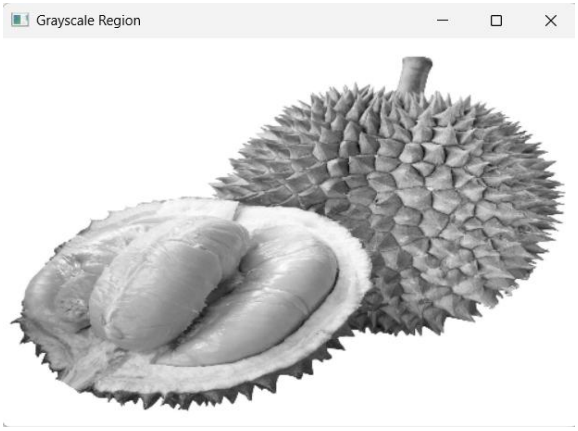

Figure 14: Grayscale Image

Image Enhancement technique is used to transform the image into grayscale. This is the main component of the background removal system as it reduces the complexity in isolate the background and the foreground.

iii. Image Restoration

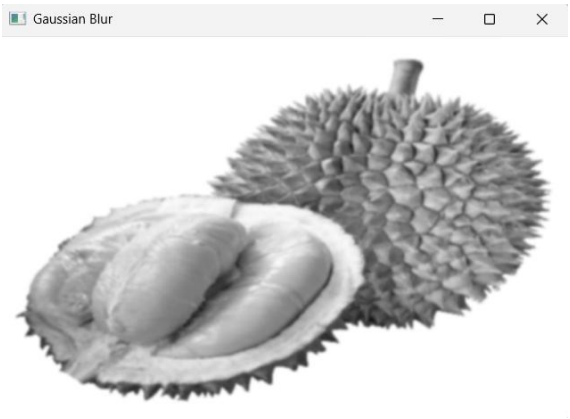

Figure 15: Image applied with Gaussian blur

Gaussian blur is applied to reduce the details in the foreground. This step minimizes excessive white regions in the foreground, creating a smoother transition for next processing which is also the main components of the system. The kernel size used for the Gaussian filter is 5x5 and OpenCV will the determine the value of the standard deviation.

iv. Image Enhancement (Histogram Equalization)

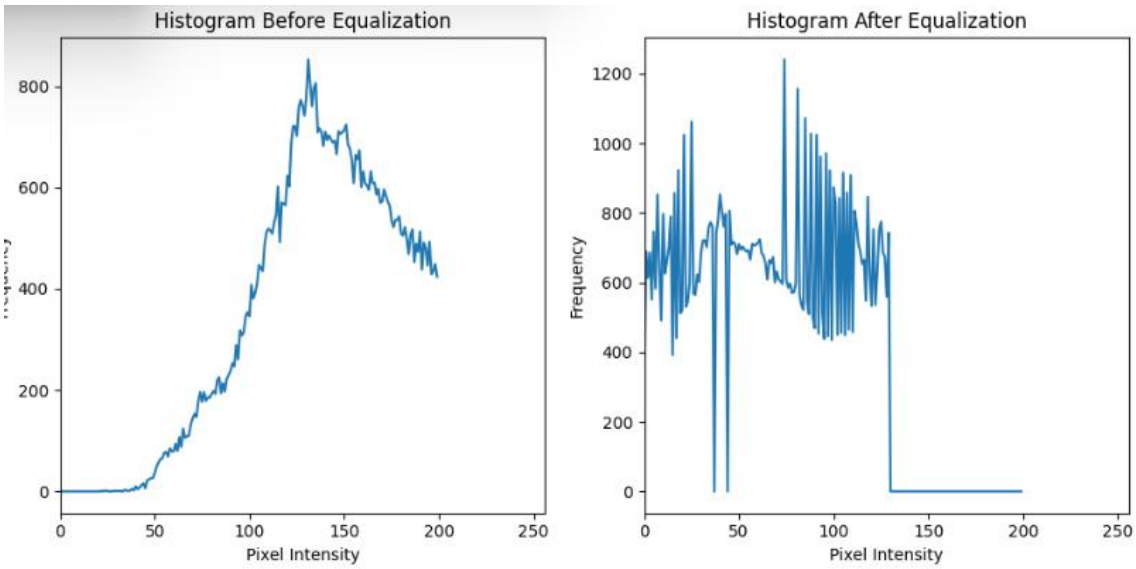

Figure 16: Histogram of before and after equalization

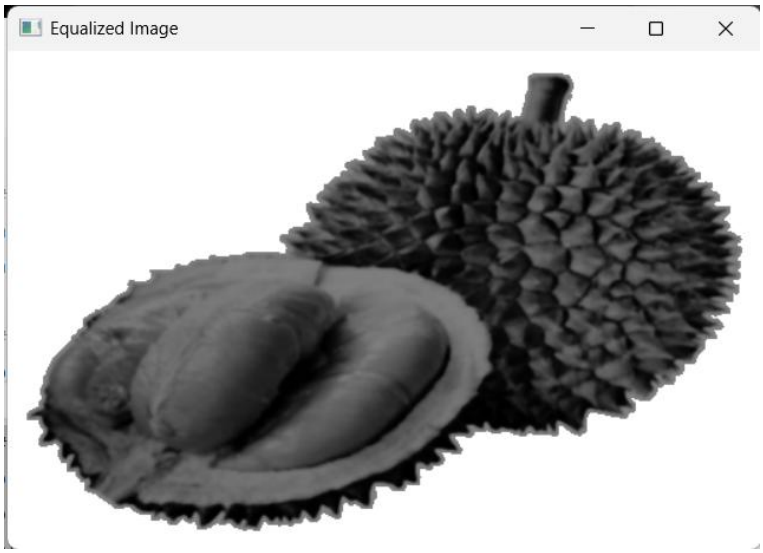

Figure 17: Equalized image

Histogram equalization is used to enhance the contrast for distributing the intensity range in foreground. The graph in figure 17 shows histogram of frequence intensity of the pixel before and after equalization. The intensity graph is cut to 200 pixels only as there is excessive frequency of pixel intensity above 250 which causing the lower intensity not visible in graph.

v. Segmentation

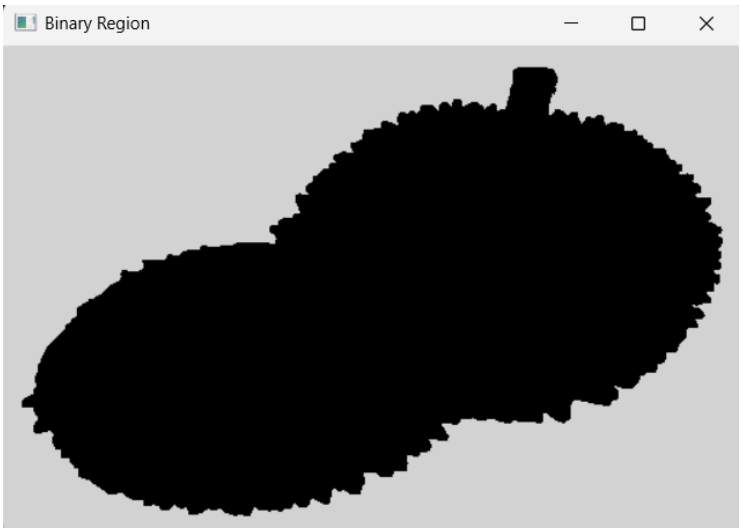

Figure 18: Binary Image.

Figure 19: Inverted Binary Image.

In segmentation process, thresholding is used to change the image to a binary image. This step is used to further simplify isolating the object of interest with the background. For the threshold setting, any pixel value above 190 will be set to white while below will be set to black. Next, the image will be inverted changing the black to white and the white to black.

vi. Colour Image Processing

Figure 20: Removed Background Image

The next process involves colour image processing, which is BGRA, where an alpha channel is used to enable transparency in the image. The code divided the channel to only effect the desire colour where white pixels set to fully opaque while the black pixels are set to fully transparent. The alpha mask is then applied to the colour channels of the selected region. The processed colour channels and the alpha channel than merged to create a new image in BGRA format.

vii. Image Compression

Lastly, the image is compressed to optimize the storage of the processed image. OpenCV have built in library to determine which method to use to compress the image. As for this result, the image is saved in PNG file, so compression is achieved using Deflate which is the combination of LZ77 algorithm and Huffman coding. PNG compression is lossless where the image's original quality and data are fully preserved.

Each of the technique contributing a necessary processing to obtain the output where missing or skipping either one might badly affect the output. This result shows how well the techniques applied in processing the images and achieved the background removal system.

4.0 CHALLENGES AND SOLUTIONS

During the development and testing of the image processing application, several challenges were encountered. Below, these challenges are outlined along with the solutions implemented to address them:

Challenge: Adding Transparency to Processed Regions

Initially, the goal was to enhance and process selected regions of the image, including adding transparency. Common color models like RGB, HSV/HIS, and YCbCr lack a dedicated transparency or alpha channel, making them unsuitable for tasks regarding controlled transparency.

Solution: To address this, the BGRA color model was used, as it includes an alpha channel (A) specifically for transparency. The alpha channel allowed precise control over the visibility of the processed region. By creating an alpha mask from the processed binary image, able to seamlessly integrate transparency into the final processed image. This approach ensured compatibility with image formats that support alpha channels, such as PNG while maintaining visual clarity.

Challenge: Image Processing Pipeline Stability

The multi-step image processing pipeline involves grayscale conversion, Gaussian blur, histogram equalization, thresholding, and binary inversion – introduced potential issues such as overly aggressive thresholding or loss of detail.

Solution: Each step of the pipeline was fine-tuned to balance processing and detail preservation. Specific parameters, such as the kernel size for the Gaussian blur and threshold values, were carefully selected after iterative testing. This iterative approach ensured that the processed image retained sufficient detail and was visually content with users' expectation.

Challenge: Compatibility with Multiple Image Formats

The application needed to handle various image formats like PNG, JPG, JPEG, and BMP. While OpenCV supports these formats, certain operations, such as handling transparency, are format-dependant. Jpeg, for example, does not support transparency.

Solution: To maintain compatibility, the application allowed input from various formats but ensured the output was always saved in the PNG format, which supports transparency. Clear

error messages and user guidance were added to handle scenarios where unsupported formats were encountered.

The first limitation of the project is its reliance on white backgrounds. The image processing pipeline, particularly the thresholding and binary inversion steps, depends on a clear contrast between the object and its background. In images with a white background, the background has high-intensity values in grayscale, making it easier to distinguish from the object. However, for images with non-white or complex backgrounds, the intensity values of the object and background may overlap, causing the thresholding process to misclassify pixels. This leads to inaccurate alpha mask generation and inconsistent transparency effects, limiting the application's ability to handle images with varying or intricate backgrounds.

Another limitation is the manual region selection process. The application requires users to draw a rectangle to specify the region of interest for processing. While this approach is straightforward for rectangular or uniformly shaped objects, it can be imprecise for selecting irregularly shaped or intricate regions. As a result, some areas of interest may be excluded, or unwanted regions may be included in the processing. This limitation highlights the need for automated region selection algorithms, such as object detection models, to improve accuracy and usability.

5.0 CONCLUSIONS AND FUTURE WORKS

This project of Background Removal System Using Digital Image Processing project successfully demonstrates the applications of various digital image processing techniques to achieve efficient and accurate background removal. The system effectively isolates and removes backgrounds from digital images by integrating geometric transformations, image enhancement, image restoration, image segmentation, colour image processing and compression. The results show that the system can handle different types of images and produce high-quality outputs suitable for various applications including photography, graphic design and computer vision.

Future work on this project could include integrating advanced algorithms for better handling of complex backgrounds. Machine learning models such as object detection and segmentation could automate the region selection process and improve accuracy for

irregularly shaped objects. Finally, enhancing the system to process images with varying background colours and patterns will expand its usability.

6.0 REFERENCES

[1] R. C. Gonzalez and R. E. Woods, "Digital Image Processing," 4th ed., Pearson, 2018.

[2] A. K. Jain, "Fundamentals of Digital Image Processing," Prentice-Hall, 1989.

[3] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, Digital Image Processing Using MATLAB, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2020.

[4] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," Computer Vision, Graphics, and Image Processing, vol. 30, no. 1, pp. 32–46, Apr. 1985, doi: 10.1016/0734-189X(85)90016-7.

[5] N. Otsu, "A threshold selection method from gray-level histograms," IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62–66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076.

7.0 ATTACHMENTS

BACKGROUND REMOVAL SYSTEM USING DIP > BackgroundRemoval.py > ...

```python
import cv2
import numpy as np

# Initialize variables for drawing
ix, iy, drawing = -1, -1, False
bx, by = -1, -1
img = cv2.imread(r"e:\DEGREE\YEAR 4\BERR4723_Digital Image Processing\Assignment\apple-banana-cherry.jpg")

if img is None:
    print("Error: Image not found or unable to load.")
else:
    # Resize the image (optional step)
    new_width = 600  # Set desired width
    new_height = 400  # Set desired height
    img = cv2.resize(img, (new_width, new_height))

    # Convert to BGRA format (with an alpha channel)
    b_channel, g_channel, r_channel = cv2.split(img)
    alpha_channel = np.ones(b_channel.shape, dtype=b_channel.dtype) * 255  # Fully opaque
    img_bgra = cv2.merge((b_channel, g_channel, r_channel, alpha_channel))  # Merge BGR channels with alpha
```

```python


# Function to draw the bounding box
def draw_rectangle(event, x, y, flags, param):
    global ix, iy, drawing, img, bx, by
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            img_copy = img.copy()
            cv2.rectangle(img_copy, (ix, iy), (x, y), (0, 255, 0), 2)
            cv2.imshow("Image", img_copy)
    elif event == cv2.EVENT_LBUTTONUP:
        drawing = False
        bx, by = x, y  # Save the final rectangle's bottom-right corner
        cv2.rectangle(img, (ix, iy), (bx, by), (0, 255, 0), 2)
        cv2.imshow("Image", img)

# Display the image and set up mouse callback for selection
cv2.imshow("Image", img)
cv2.setMouseCallback("Image", draw_rectangle)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Ensure selection is valid
if ix != -1 and iy != -1:
    # Step 1: Adjust the coordinates to remove the border (optional border thickness can be subtracted)
    border_thickness = 2  # Adjust the border thickness if needed
    selected_region = img[iy + border_thickness:by - border_thickness, ix + border_thickness:bx - border_thickness]

    # Step 2: Convert only the selected region to grayscale
    gray_region = cv2.cvtColor(selected_region, cv2.COLOR_BGR2GRAY)
    cv2.imshow("Grayscale Region", gray_region)  # Show grayscale image
    cv2.waitKey(0)

    # Step 3: Apply Gaussian blur to reduce noise
    blurred_region = cv2.GaussianBlur(gray_region, (5, 5), 0)

    # Step 3.1: Apply histogram equalization to make the object darker (enhance contrast)
    equalized_region = cv2.equalizeHist(blurred_region)
    cv2.imshow("Equalized Region", equalized_region)  # Show equalized image
    cv2.waitKey(0)

    # Step 4: Perform thresholding to create a binary image (only white and black)
    _, binary_region = cv2.threshold(equalized_region, 190, 210, cv2.THRESH_BINARY)
    cv2.imshow("Binary Region", binary_region)  # Show binary image
    cv2.waitKey(0)

    # Step 5: Invert the binary image (white to black and black to white)
    inverted_region = cv2.bitwise_not(binary_region)
    cv2.imshow("Inverted Region", inverted_region)  # Show inverted image
    cv2.waitKey(0)

    # Step 6: Create an alpha mask where white is 0 (transparent) and black is 255 (opaque)
    alpha_mask = inverted_region  # This mask now ensures the object is removed (transparent)
    alpha_mask = alpha_mask.astype(np.uint8)

    # Step 7: Apply the alpha mask to the color channels (B, G, R) of the selected region
    for c in range(3):  # Iterate over color channels (0: Blue, 1: Green, 2: Red)
        selected_region[:, :, c] = selected_region[:, :, c] * (alpha_mask / 255.0)

    # Step 8: Create a new image (BGRA) for saving the selected region
    selected_region_bgra = cv2.merge((selected_region[:, :, 0], selected_region[:, :, 1], selected_region[:, :, 2], alpha_mask))

    # Step 9: Compress and save the selected region with transparency as a PNG file
    png_compression_params = [cv2.IMWRITE_PNG_COMPRESSION, 5]  # Level 5 compression (out of 0-9)
    cv2.imwrite("Cropped_PNG_Image.png", selected_region_bgra, png_compression_params)

    # Display the saved PNG image with transparency
    cv2.imshow("Saved PNG Image", selected_region_bgra)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Figure 21: Codes of the system