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.

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. This comprehensive approach makes the system versatile and efficient which is suitable for applications in photography, graphic design, and computer vision.

2.0 METHODOLOGY

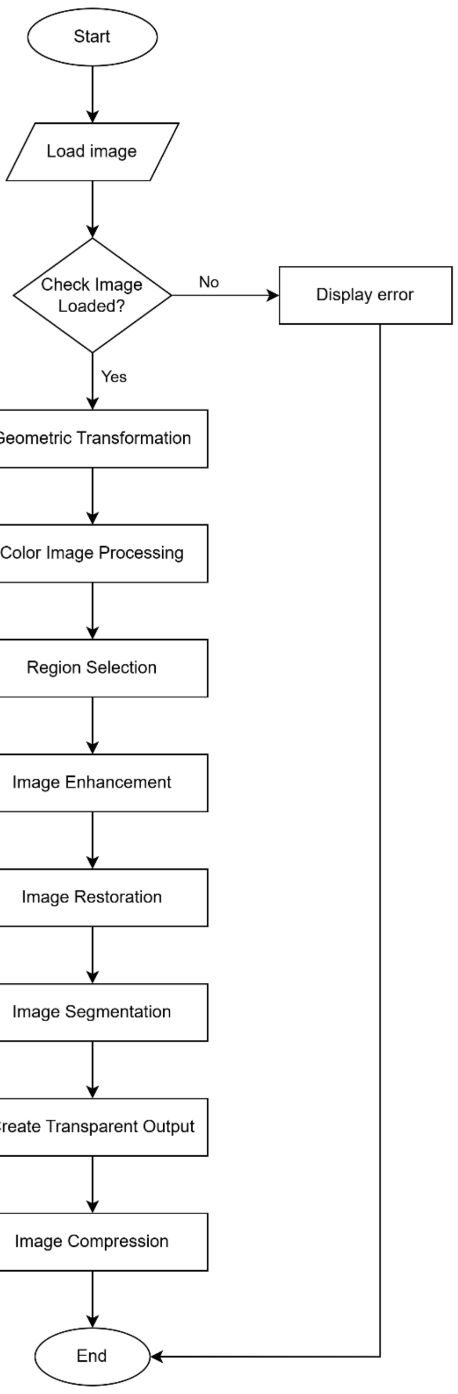
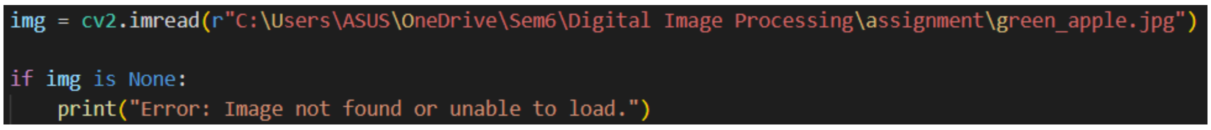


Figure 1: Flowchart of the system

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.

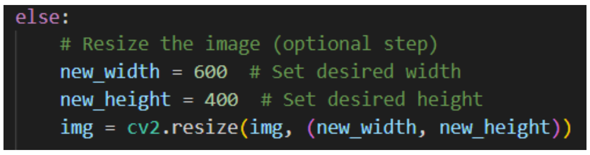


```python
img = cv2.imread(r"C:\Users\ASUS\OneDrive\Sem6\Digital Image Processing\assignment\green_apple.jpg")

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.

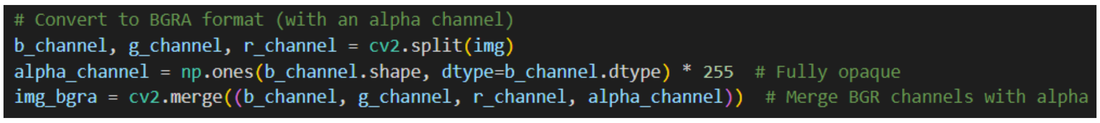


```python
else:
    # Resize the image (optional step)
    new_width = 600  # Set desired width
    new_height = 400  # Set desired height
    img = cv2.resize(img, (new_width, new_height))
```

Figure 3: Code to resize the image

iii. Color Image Processing

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

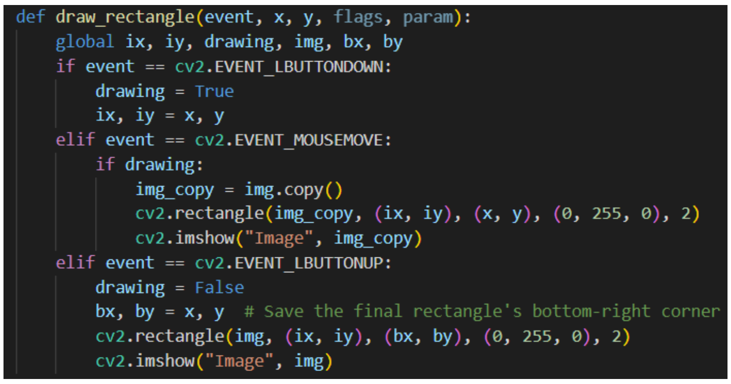


```python
# Convert to BGRA format (with an alpha channel)
b_channel, g_channel, r_channel = cv2.split(img)
alpha_channel = np.ones(b_channel.shape, dtype=b_channel.dtype) * 255  # Fully opaque
img_bgra = cv2.merge((b_channel, g_channel, r_channel, alpha_channel))  # Merge BGR channels with alpha
```

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).

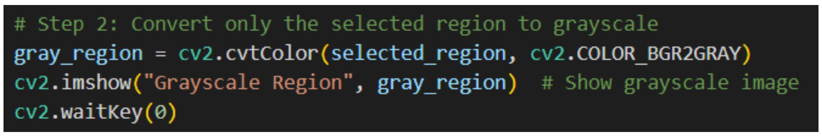


```python
def draw_rectangle(event, x, y, flags, param):
    global ix, iy, drawing, img, bx, by
    if event == cv2.EVENT_LBUTTONDOWN:
        drawing = True
        ix, iy = x, y
    elif event == cv2.EVENT_MOUSEMOVE:
        if drawing:
            img_copy = img.copy()
            cv2.rectangle(img_copy, (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

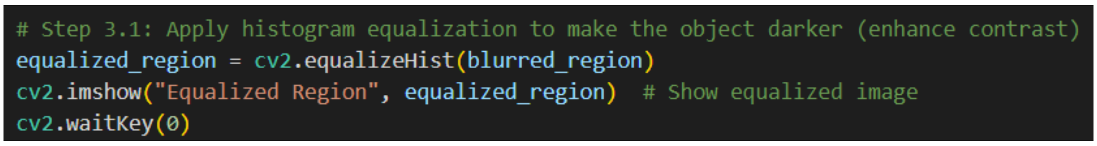
Then, the image with the selected region is converted to grayscale to focus on intensity values. Besides, grayscale is essential for tasks like histogram equalization and thresholding as they rely on intensity values.



```
# 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.

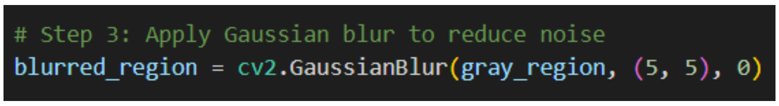


```
# Step 3.1: Apply histogram equalization to make the object darker (enhance contrast)
equalized_region = cv2.equalizeHist(blurred_region)
cv2.imshow("Equalized Region", equalized_region)  # Show equalized image
cv2.waitKey(0)
```

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.

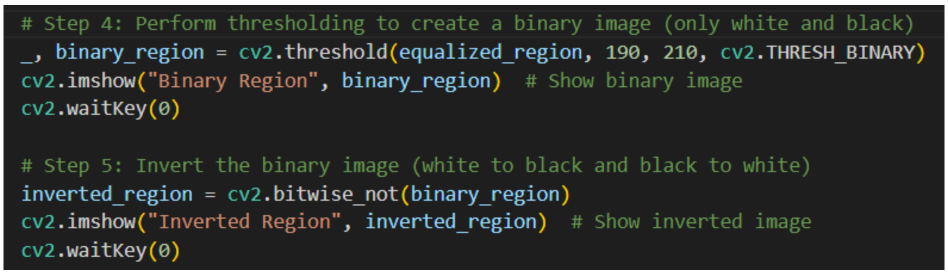


```
# Step 3: Apply Gaussian blur to reduce noise
blurred_region = cv2.GaussianBlur(gray_region, (5, 5), 0)
```

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.

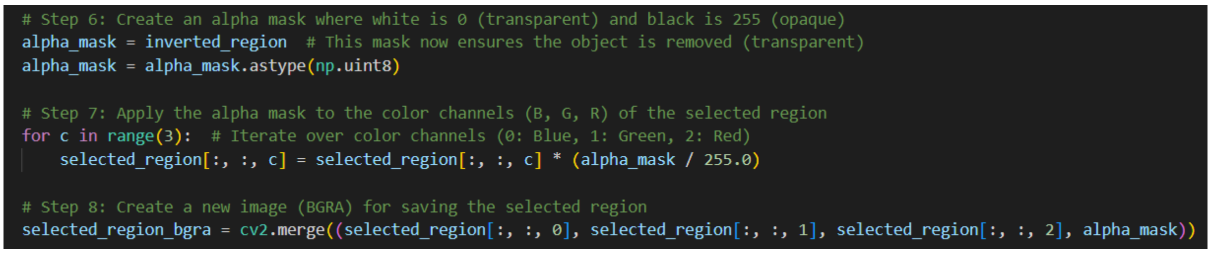


```
# Step 4: Perform thresholding to create a binary image (only white and black)
_, binary_region = cv2.threshold(equalized_region, 190, 210, cv2.THRESH_BINARY)
cv2.imshow("Binary Region", binary_region)  # Show binary image
cv2.waitKey(0)

# 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.

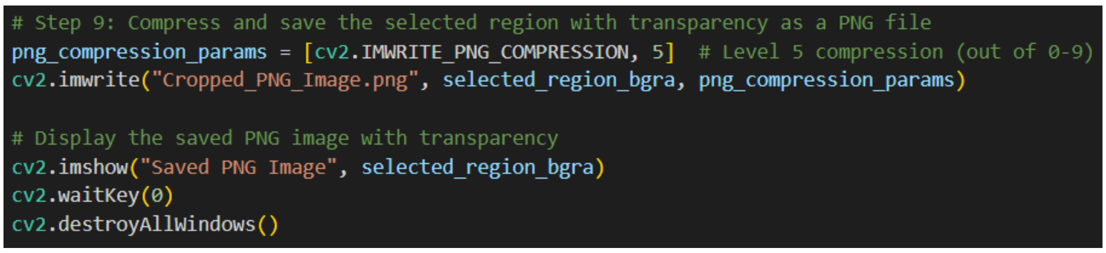


```python
# Step 6: Create an alpha mask where white is 0 (transparent) and black is 255 (opaque)
alpha_mask = inverted_region  # This mask now ensures the object is removed (transparent)
alpha_mask = alpha_mask.astype(np.uint8)

# 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.



```python
# 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

4.0 CHALLENGES AND SOLUTIONS

5.0 CONCLUSIONS AND FUTURE WORKS