SRM Institute of Science and Technology

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Department of Computer Applications

Circular – 2023-24

B.Sc. CS 6th Sem

COMPUTER VISION FUNDAMENTALS (UCS23G04J)- Lab Manual

This manual provides a structured guide for each program listed, covering the aim, procedure, source code, example input, and expected output.

Lab 1: Install OpenCV & Displaying Images

Title

Installing OpenCV and Displaying Images

Aim

To install the OpenCV library in Python and write a program to load and display an image.

Procedure

1. **Install OpenCV:** Open your terminal or command prompt and run the following command to install `opencv-python` and `numpy`:
2. `pip install opencv-python numpy`

3. **Prepare an Image:** Ensure you have an image file (e.g., `example.jpg`) in the same directory as your Python script, or provide the full path to the image.
4. **Write the Python Code:** Create a new Python file (e.g., `lab1.py`) and add the source code provided below.
5. **Run the Script:** Execute the Python script from your terminal:
6. `python lab1.py`

7. **Observe Output:** A window titled "Displayed Image" should appear, showing your image. Press any key to close the window.

Source Code

```
import cv2

def display_image(image_path):
    """
    Loads and displays an image using OpenCV.

    Args:
        image_path (str): The path to the image file.
    """
```

```
    # Read the image from the specified path
    img = cv2.imread(image_path)

    # Check if the image was loaded successfully
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the image in a window
    cv2.imshow('Displayed Image', img)

    # Wait indefinitely until a key is pressed (0 means wait forever)
    cv2.waitKey(0)

    # Destroy all OpenCV windows
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace 'path/to/your/image.jpg' with the actual path to your image
file
    image_file = 'path/to/your/image.jpg'
    display_image(image_file)
```

Input

A valid path to an image file, e.g., `'my_image.jpg'`

Expected Output

A new window titled "Displayed Image" will open, displaying the content of `my_image.jpg`. The window will close when any key is pressed.

Lab 2: Reading & Writing Images

Title

Reading and Writing Images with OpenCV

Aim

To read an image from a specified path and then save it to a new file, potentially in a different format, using OpenCV.

Procedure

1. **Prepare an Image:** Have an input image file (e.g., `input.png`).
2. **Write the Python Code:** Create a Python file (e.g., `lab2.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab2.py`

5. **Verify Output:** Check the directory where you ran the script for the newly saved image file (e.g., `output.jpg`).

Source Code

```
import cv2

def read_and_write_image(input_path, output_path):
    """
    Reads an image from input_path and writes it to output_path.

    Args:
        input_path (str): The path to the input image file.
        output_path (str): The path where the image will be saved.
    """
    # Read the image
    img = cv2.imread(input_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {input_path}")
        return

    # Write the image to the specified output path
    # The file extension in output_path determines the format (e.g., .jpg,
.png)
    success = cv2.imwrite(output_path, img)

    if success:
        print(f"Image successfully read from '{input_path}' and written to
'{output_path}'")
    else:
        print(f"Error: Could not write image to '{output_path}'")

if __name__ == "__main__":
    # Replace with your input and desired output paths
    input_image_file = 'path/to/your/input.png'
    output_image_file = 'output_image.jpg' # Can change extension to .png,
.bmp, etc.
    read_and_write_image(input_image_file, output_image_file)
```

Input

`input_path`: `'path/to/your/input.png'` (an existing image file) `output_path`: `'output_image.jpg'` (a new file name for the saved image)

Expected Output

A message indicating successful reading and writing, and a new image file named `output_image.jpg` (or whatever `output_path` specifies) will be created in the execution directory.

Lab 3: Draw a Rectangle & Draw a Circle

Title

Drawing Basic Shapes (Rectangle and Circle) on an Image

Aim

To learn how to draw a rectangle and a circle on an existing image using OpenCV's drawing functions.

Procedure

1. **Prepare an Image:** Use an image file (e.g., `blank_canvas.jpg` or any other image).
2. **Write the Python Code:** Create a Python file (e.g., `lab3.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab3.py`

5. **Observe Output:** A window titled "Image with Shapes" will appear, displaying the image with a drawn rectangle and circle.

Source Code

```
import cv2
import numpy as np

def draw_shapes_on_image(image_path):
    """
    Loads an image and draws a rectangle and a circle on it.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # If image not found, create a blank white image for demonstration
    if img is None:
        print(f"Warning: Could not load image from {image_path}. Creating a
blank image.")
        img = np.zeros((500, 500, 3), dtype=np.uint8) # Create a 500x500
black image
        img[:] = (255, 255, 255) # Make it white

    # Define rectangle parameters: (top-left corner), (bottom-right corner),
color, thickness
    # Color is BGR (Blue, Green, Red)
    start_point_rect = (50, 50)
    end_point_rect = (200, 200)
    color_rect = (255, 0, 0) # Blue color
    thickness_rect = 2 # Pixels

    # Draw the rectangle
    cv2.rectangle(img, start_point_rect, end_point_rect, color_rect,
thickness_rect)

    # Define circle parameters: center coordinates, radius, color, thickness
    center_coordinates_circle = (350, 350)
    radius_circle = 70
```

```python
    color_circle = (0, 255, 0) # Green color
    thickness_circle = -1 # -1 means fill the circle

    # Draw the circle
    cv2.circle(img, center_coordinates_circle, radius_circle, color_circle, 
thickness_circle)

    # Display the image with shapes
    cv2.imshow('Image with Shapes', img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path or let it create a blank image
    image_file = 'path/to/your/image.jpg'
    draw_shapes_on_image(image_file)
```

Input

A valid path to an image file, e.g., `'my_canvas.png'`. If the image is not found, a blank white image will be created.

Expected Output

A new window titled "Image with Shapes" will open, displaying the input image (or a blank white image) with a blue rectangle and a filled green circle drawn on it.

Lab 4: Text in Images

Title

Adding Text to Images

Aim

To add custom text strings to an image at specified locations using OpenCV's text rendering functions.

Procedure

1. **Prepare an Image:** Use an image file (e.g., `background.jpg`).
2. **Write the Python Code:** Create a Python file (e.g., `lab4.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab4.py`

5. **Observe Output:** A window titled "Image with Text" will appear, showing the image with the added text.

Source Code

```
import cv2
import numpy as np

def add_text_to_image(image_path, text_to_add):
    """
    Loads an image and adds text to it.

    Args:
        image_path (str): The path to the image file.
        text_to_add (str): The string of text to add.
    """
    # Read the image
    img = cv2.imread(image_path)

    # If image not found, create a blank white image for demonstration
    if img is None:
        print(f"Warning: Could not load image from {image_path}. Creating a 
blank image.")
        img = np.zeros((400, 600, 3), dtype=np.uint8) # Create a 400x600 
black image
        img[:] = (255, 255, 255) # Make it white

    # Define text parameters
    font = cv2.FONT_HERSHEY_SIMPLEX
    org = (50, 50) # Bottom-left corner of the text string in the image
    font_scale = 1.2
    color = (0, 0, 255) # Red color (BGR)
    thickness = 2
    line_type = cv2.LINE_AA # Anti-aliased line for smoother text

    # Put the text on the image
    cv2.putText(img, text_to_add, org, font, font_scale, color, thickness, 
line_type)

    # Display the image with text
    cv2.imshow('Image with Text', img)
```

```
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path and desired text
    image_file = 'path/to/your/image.jpg'
    my_text = 'Hello, OpenCV!'
    add_text_to_image(image_file, my_text)
```

Input

`image_path`: `'path/to/your/image.jpg'` `text_to_add`: `'Computer Vision Lab'`

Expected Output

A new window titled "Image with Text" will open, displaying the input image (or a blank white image) with the text "Computer Vision Lab" written in red at the top-left corner.

Lab 5: Color Space OpenCV & Thresholding OpenCV

Title

Color Space Conversion and Image Thresholding

Aim

To convert an image between different color spaces (e.g., BGR to Grayscale, BGR to HSV) and apply binary thresholding to segment the image.

Procedure

1. **Prepare an Image:** Use a color image file (e.g., `color_image.jpg`).
2. **Write the Python Code:** Create a Python file (e.g., `lab5.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab5.py`

5. **Observe Output:** Three windows will appear: the original image, its grayscale version, its HSV version, and its thresholded (binary) version.

Source Code

```
import cv2

def color_space_and_thresholding(image_path):
    """
    Loads a color image, converts it to grayscale and HSV,
    and applies binary thresholding.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # Convert BGR image to Grayscale
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Grayscale Image', gray_img)

    # Convert BGR image to HSV color space
    hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    cv2.imshow('HSV Image', hsv_img)

    # Apply binary thresholding to the grayscale image
    # Pixels with intensity > 127 become 255 (white), others become 0 (black)
    ret, thresh_img = cv2.threshold(gray_img, 127, 255, cv2.THRESH_BINARY)
    cv2.imshow('Thresholded Image (Binary)', thresh_img)

    # Wait for a key press and then close all windows
    cv2.waitKey(0)
```

```
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path
    image_file = 'path/to/your/color_image.jpg'
    color_space_and_thresholding(image_file)
```

Input

A valid path to a color image file, e.g., `'my_color_photo.jpg'`.

Expected Output

Four separate windows will open:

1. "Original Image": Displays the input color image.
2. "Grayscale Image": Displays the grayscale version of the input image.
3. "HSV Image": Displays the input image converted to HSV color space.
4. "Thresholded Image (Binary)": Displays a black and white image where pixels above a certain intensity in the grayscale image are white, and others are black.

Lab 6: Finding Contours

Title

Finding and Drawing Contours in an Image

Aim

To detect and draw contours (outlines of objects) present in a binary image using OpenCV.

Procedure

1. **Prepare an Image:** Use an image with clear objects or shapes, or a simple binary image.
2. **Write the Python Code:** Create a Python file (e.g., `lab6.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab6.py`

5. **Observe Output:** A window titled "Contours Detected" will appear, showing the image with the detected contours drawn in green.

Source Code

```
import cv2
import numpy as np

def find_and_draw_contours(image_path):
    """
    Loads an image, converts it to grayscale, applies thresholding,
    finds contours, and draws them on the original image.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Convert the image to grayscale
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Apply binary thresholding to get a binary image
    # This is crucial for contour detection. Adjust threshold value if
needed.
    ret, thresh_img = cv2.threshold(gray_img, 100, 255, cv2.THRESH_BINARY)
    cv2.imshow('Thresholded for Contours', thresh_img)

    # Find contours in the binary image
    # cv2.RETR_EXTERNAL retrieves only the extreme outer contours
    # cv2.CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal
segments
    contours, hierarchy = cv2.findContours(thresh_img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Draw all found contours on the original image
    # -1 means draw all contours, (0, 255, 0) is green color, 2 is thickness
    cv2.drawContours(img, contours, -1, (0, 255, 0), 2)
```

```
    # Display the image with drawn contours
    cv2.imshow('Contours Detected', img)

    # Wait for a key press and then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path (an image with distinct objects works
best)
    image_file = 'path/to/your/shapes_image.png'
    find_and_draw_contours(image_file)
```

Input

A valid path to an image file, e.g., `'shapes.png'`, which contains clear objects or shapes.

Expected Output

Two windows will open:

1. "Thresholded for Contours": Displays the binary version of the input image, which is used for contour detection.
2. "Contours Detected": Displays the original input image with green outlines drawn around the detected contours of objects.

Lab 7: Image Edge Detection OpenCV

Title

Image Edge Detection using OpenCV

Aim

To apply various edge detection algorithms, specifically the Canny edge detector, to an image to identify significant changes in image intensity.

Procedure

1. **Prepare an Image:** Use an image with clear features or objects (e.g., `building.jpg`).
2. **Write the Python Code:** Create a Python file (e.g., `lab7.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab7.py`

5. **Observe Output:** Two windows will appear: the original image and its Canny edge-detected version.

Source Code

```
import cv2

def image_edge_detection(image_path):
    """
    Loads an image, converts it to grayscale, and applies Canny edge
detection.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # Convert the image to grayscale (Canny operates on grayscale images)
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Apply Canny edge detector
    # Arguments: (image, threshold1, threshold2)
    # Any gradient value larger than threshold2 is considered an edge.
    # Any gradient value smaller than threshold1 is considered not an edge.
    # Gradient values between threshold1 and threshold2 are considered edges
    # if they are connected to "sure-edge" pixels.
    edges = cv2.Canny(gray_img, 100, 200)

    # Display the edge-detected image
    cv2.imshow('Canny Edges', edges)

    # Wait for a key press and then close all windows
```

```
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path
    image_file = 'path/to/your/image_with_edges.jpg'
    image_edge_detection(image_file)
```

Input

A valid path to an image file, e.g., `'landscape.jpg'`.

Expected Output

Two windows will open:

1. "Original Image": Displays the input image.
2. "Canny Edges": Displays a black and white image where the detected edges are shown in white against a black background.

Lab 8: Image Scaling & Rotation using OpenCV

Title

Image Scaling and Rotation

Aim

To perform image scaling (resizing) and rotation operations using OpenCV functions, demonstrating affine transformations.

Procedure

1. **Prepare an Image:** Use any image file (e.g., `photo.jpg`).
2. **Write the Python Code:** Create a Python file (e.g., `lab8.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab8.py`

5. **Observe Output:** Three windows will appear: the original image, a scaled version, and a rotated version.

Source Code

```
import cv2
import numpy as np

def image_scaling_and_rotation(image_path):
    """
    Loads an image, scales it up/down, and rotates it.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # --- Image Scaling ---
    # Define scaling factors
    scale_percent = 50 # 50% of original size (downscale)
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)

    # Resize image using INTER_AREA for shrinking, INTER_LINEAR for zooming
    scaled_img = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
    cv2.imshow('Scaled Image (50%)', scaled_img)

    # --- Image Rotation ---
    # Get image dimensions
    (h, w) = img.shape[:2]
    center = (w // 2, h // 2)
```

```python
    # Define rotation parameters: center, angle, scale
    angle = 45 # Rotate by 45 degrees
    scale = 1.0 # No scaling during rotation

    # Get the 2D rotation matrix
    M = cv2.getRotationMatrix2D(center, angle, scale)

    # Perform the rotation
    # The third argument is the output image size (width, height)
    rotated_img = cv2.warpAffine(img, M, (w, h))
    cv2.imshow('Rotated Image (45 degrees)', rotated_img)

    # Wait for a key press and then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path
    image_file = 'path/to/your/sample_image.jpg'
    image_scaling_and_rotation(image_file)
```

Input

A valid path to an image file, e.g., `'cityscape.jpg'`.

Expected Output

Three windows will open:

1. "Original Image": Displays the input image.
2. "Scaled Image (50%)": Displays the image resized to 50% of its original dimensions.
3. "Rotated Image (45 degrees)": Displays the image rotated by 45 degrees around its center.

Lab 9: Image Translation OpenCV & Image Filtering OpenCV

Title

Image Translation and Basic Image Filtering

Aim

To perform image translation (shifting) and apply a basic custom 2D convolution filter to an image.

Procedure

1. **Prepare an Image:** Use any image file (e.g., `texture.png`).
2. **Write the Python Code:** Create a Python file (e.g., `lab9.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab9.py`

5. **Observe Output:** Three windows will appear: the original image, a translated version, and a filtered version.

Source Code

```
import cv2
import numpy as np

def image_translation_and_filtering(image_path):
    """
    Loads an image, translates it, and applies a custom 2D filter.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # --- Image Translation ---
    # Define translation values (shift 100 pixels right, 50 pixels down)
    tx, ty = 100, 50
    # Create the 2x3 translation matrix
    M_translate = np.float32([[1, 0, tx], [0, 1, ty]])
    # Get image dimensions
    (h, w) = img.shape[:2]

    # Apply the translation
    translated_img = cv2.warpAffine(img, M_translate, (w, h))
    cv2.imshow('Translated Image', translated_img)

    # --- Image Filtering (2D Convolution) ---
    # Define a custom 3x3 kernel (e.g., a simple sharpening filter)
```

```
    # Note: Sum of kernel elements should ideally be 1 for brightness
preservation
    # or 0 for edge detection/sharpening without changing overall brightness.
    kernel = np.array([[-1, -1, -1],
                       [-1,  9, -1],
                       [-1, -1, -1]], dtype=np.float32)

    # Apply the 2D convolution filter
    # -1 indicates that the depth of the output image will be the same as the
input
    filtered_img = cv2.filter2D(img, -1, kernel)
    cv2.imshow('Filtered Image (Sharpening)', filtered_img)

    # Wait for a key press and then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path
    image_file = 'path/to/your/another_image.jpg'
    image_translation_and_filtering(image_file)
```

Input

A valid path to an image file, e.g., `'photo_for_filter.jpg'`.

Expected Output

Three windows will open:

1. "Original Image": Displays the input image.
2. "Translated Image": Displays the image shifted by 100 pixels to the right and 50 pixels down.
3. "Filtered Image (Sharpening)": Displays the image after applying a basic sharpening filter.

Lab 10: Image Filtering Blurring OpenCV & Image Filtering Blurring Gaussian Blur OpenCV

Title

Image Blurring (Average and Gaussian)

Aim

To apply two common blurring techniques – average blurring and Gaussian blurring – to an image using OpenCV, understanding their effects on image smoothness.

Procedure

1. **Prepare an Image:** Use an image that could benefit from blurring (e.g., `noisy_image.jpg`).
2. **Write the Python Code:** Create a Python file (e.g., `lab10.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab10.py`

5. **Observe Output:** Three windows will appear: the original image, an average blurred version, and a Gaussian blurred version.

Source Code

```
import cv2
import numpy as np

def image_blurring(image_path):
    """
    Loads an image and applies average and Gaussian blurring.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # --- Average Blurring ---
    # Define kernel size (e.g., 5x5)
    # The larger the kernel, the more blurred the image will be.
    kernel_size_avg = (5, 5)
    average_blurred_img = cv2.blur(img, kernel_size_avg)
    cv2.imshow('Average Blurred Image (5x5)', average_blurred_img)

    # --- Gaussian Blurring ---
    # Define kernel size (e.g., 5x5) and standard deviation in X and Y
direction (sigmaX, sigmaY)
    # sigmaX = 0 means it's calculated automatically based on kernel size.
    kernel_size_gaussian = (5, 5)
```

```
    gaussian_blurred_img = cv2.GaussianBlur(img, kernel_size_gaussian, 0)
    cv2.imshow('Gaussian Blurred Image (5x5)', gaussian_blurred_img)

    # Wait for a key press and then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path
    image_file = 'path/to/your/image_to_blur.jpg'
    image_blurring(image_file)
```

Input

A valid path to an image file, e.g., `'detailed_photo.jpg'`.

Expected Output

Three windows will open:

1. "Original Image": Displays the input image.
2. "Average Blurred Image (5x5)": Displays the image after applying an average blur with a 5x5 kernel.
3. "Gaussian Blurred Image (5x5)": Displays the image after applying a Gaussian blur with a 5x5 kernel.

Lab 11: Image Filtering Blurring Median Blur OpenCV & Morphological Operations Erosion OpenCV

Title

Median Blurring and Morphological Erosion

Aim

To apply median blurring for noise reduction and perform morphological erosion, which shrinks foreground objects.

Procedure

1. **Prepare an Image:** Use an image, preferably one with salt-and-pepper noise for median blur, and a binary image with distinct foreground objects for erosion.
2. **Write the Python Code:** Create a Python file (e.g., `lab11.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab11.py`

5. **Observe Output:** Three windows will appear: the original image, a median blurred version, and an eroded version.

Source Code

```
import cv2
import numpy as np

def median_blur_and_erosion(image_path):
    """
    Loads an image, applies median blurring, and performs morphological
erosion.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # --- Median Blurring ---
    # Useful for removing salt-and-pepper noise. Kernel size must be odd.
    median_blurred_img = cv2.medianBlur(img, 5) # 5x5 kernel
    cv2.imshow('Median Blurred Image (5x5)', median_blurred_img)

    # --- Morphological Erosion ---
    # Erosion shrinks foreground objects (white pixels) and increases the
size of background (black pixels).
    # It is useful for removing small white noises (salt noise) or detaching
two joined objects.
```

```
    # Create a structuring element (kernel)
    kernel = np.ones((5,5), np.uint8) # 5x5 square kernel of ones

    # Apply erosion
    # iterations specifies how many times erosion is applied
    eroded_img = cv2.erode(img, kernel, iterations=1)
    cv2.imshow('Eroded Image', eroded_img)

    # Wait for a key press and then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path. For erosion, a binary image or an image
    # with clear foreground/background is ideal.
    image_file = 'path/to/your/noisy_or_binary_image.png'
    median_blur_and_erosion(image_file)
```

Input

A valid path to an image file, e.g., `'noisy_text.png'` (for median blur) or `'binary_shapes.png'` (for erosion).

Expected Output

Three windows will open:

1. "Original Image": Displays the input image.
2. "Median Blurred Image (5x5)": Displays the image after applying median blur, useful for noise reduction.
3. "Eroded Image": Displays the image after morphological erosion, where foreground objects appear smaller.

Lab 12: Morphological Operations Dilation OpenCV

Title

Morphological Dilation

Aim

To apply the morphological dilation operation, which expands foreground objects in an image.

Procedure

1. **Prepare an Image:** Use a binary image with distinct foreground objects, or an image where you want to expand white regions (e.g., `thin_lines.png`).
2. **Write the Python Code:** Create a Python file (e.g., `lab12.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab12.py`

5. **Observe Output:** Two windows will appear: the original image and a dilated version.

Source Code

```
import cv2
import numpy as np

def morphological_dilation(image_path):
    """
    Loads an image and performs morphological dilation.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # --- Morphological Dilation ---
    # Dilation expands foreground objects (white pixels) and shrinks the size
of background (black pixels).
    # It is useful for joining broken parts of an object or filling small
holes.

    # Create a structuring element (kernel)
    kernel = np.ones((5,5), np.uint8) # 5x5 square kernel of ones

    # Apply dilation
    # iterations specifies how many times dilation is applied
    dilated_img = cv2.dilate(img, kernel, iterations=1)
    cv2.imshow('Dilated Image', dilated_img)

    # Wait for a key press and then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
if __name__ == "__main__":
    # Replace with your image path. A binary image or an image with clear
    # foreground/background is ideal for observing dilation effects.
    image_file = 'path/to/your/binary_image.png'
    morphological_dilation(image_file)
```

Input

A valid path to an image file, e.g., `'broken_text.png'`.

Expected Output

Two windows will open:

1. "Original Image": Displays the input image.
2. "Dilated Image": Displays the image after morphological dilation, where foreground objects appear larger or connected.

Lab 13: Image Filtering Bilateral OpenCV

Title

Bilateral Image Filtering

Aim

To apply bilateral filtering to an image, which is effective at reducing noise while preserving edges.

Procedure

1. **Prepare an Image:** Use an image where you want to reduce noise but keep sharp edges (e.g., `portrait.jpg`).
2. **Write the Python Code:** Create a Python file (e.g., `lab13.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab13.py`

5. **Observe Output:** Two windows will appear: the original image and a bilateral filtered version.

Source Code

```
import cv2

def bilateral_image_filtering(image_path):
    """
    Loads an image and applies bilateral filtering.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # --- Bilateral Filtering ---
    # Arguments:
    # d: Diameter of each pixel neighborhood.
    # sigmaColor: Filter sigma in the color space. A larger value means that
    #             farther colors within the pixel neighborhood (and farther
    #             from the center pixel in color space) will be mixed
together.
    # sigmaSpace: Filter sigma in the coordinate space. A larger value means
that
    #             farther pixels will influence each other as long as their
colors
    #             are close enough.
    # Using 75 for both sigmaColor and sigmaSpace is a common starting point.
    bilateral_filtered_img = cv2.bilateralFilter(img, 9, 75, 75)
    cv2.imshow('Bilateral Filtered Image', bilateral_filtered_img)
```

```
    # Wait for a key press and then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path. Images with noise but sharp edges are
ideal.
    image_file = 'path/to/your/noisy_portrait.jpg'
    bilateral_image_filtering(image_file)
```

Input

A valid path to an image file, e.g., `'noisy_face.jpg'`.

Expected Output

Two windows will open:

1. "Original Image": Displays the input image.
2. "Bilateral Filtered Image": Displays the image after applying bilateral filtering, showing noise reduction while preserving edges.

Lab 14: Morphological Operations Opening OpenCV

Title

Morphological Opening

Aim

To apply the morphological opening operation, which is useful for removing small objects (noise) from an image while preserving the shape and size of larger objects.

Procedure

1. **Prepare an Image:** Use a binary image that might have small noise particles (white dots) or thin connections you want to break (e.g., `noisy_binary.png`).
2. **Write the Python Code:** Create a Python file (e.g., `lab14.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab14.py`

5. **Observe Output:** Two windows will appear: the original image and an opened version.

Source Code

```
import cv2
import numpy as np

def morphological_opening(image_path):
    """
    Loads an image and performs morphological opening.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # --- Morphological Opening ---
    # Opening is an erosion followed by a dilation.
    # It removes small objects from the foreground (white noise)
    # and smoothes the contour of a foreground object.

    # Create a structuring element (kernel)
    kernel = np.ones((5,5), np.uint8) # 5x5 square kernel

    # Apply opening
    opened_img = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
    cv2.imshow('Opened Image', opened_img)

    # Wait for a key press and then close all windows
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
if __name__ == "__main__":
    # Replace with your image path. A binary image with small noise is ideal.
    image_file = 'path/to/your/binary_with_noise.png'
    morphological_opening(image_file)
```

Input

A valid path to an image file, e.g., `'binary_with_speckles.png'`.

Expected Output

Two windows will open:

1. "Original Image": Displays the input image.
2. "Opened Image": Displays the image after morphological opening, where small foreground noise particles are removed.

Lab 15: Morphological Operations Closing OpenCV

Title

Morphological Closing

Aim

To apply the morphological closing operation, which is useful for filling small holes inside foreground objects or connecting broken parts of an object.

Procedure

1. **Prepare an Image:** Use a binary image that might have small holes within foreground objects or broken connections (e.g., `broken_shape.png`).
2. **Write the Python Code:** Create a Python file (e.g., `lab15.py`) and add the source code.
3. **Run the Script:** Execute the Python script:
4. `python lab15.py`

5. **Observe Output:** Two windows will appear: the original image and a closed version.

Source Code

```
import cv2
import numpy as np

def morphological_closing(image_path):
    """
    Loads an image and performs morphological closing.

    Args:
        image_path (str): The path to the image file.
    """
    # Read the image
    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return

    # Display the original image
    cv2.imshow('Original Image', img)

    # --- Morphological Closing ---
    # Closing is a dilation followed by an erosion.
    # It is useful for filling small holes inside the foreground objects
    # and smoothing the contour of a foreground object. It also helps to
connect
    # nearby foreground objects.

    # Create a structuring element (kernel)
    kernel = np.ones((5,5), np.uint8) # 5x5 square kernel

    # Apply closing
    closed_img = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
    cv2.imshow('Closed Image', closed_img)

    # Wait for a key press and then close all windows
```

```
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    # Replace with your image path. A binary image with small holes or gaps
is ideal.
    image_file = 'path/to/your/binary_with_holes.png'
    morphological_closing(image_file)
```

Input

A valid path to an image file, e.g., `'text_with_gaps.png'`.

Expected Output

Two windows will open:

1. "Original Image": Displays the input image.
2. "Closed Image": Displays the image after morphological closing, where small holes within foreground objects are filled, and broken connections are joined.