Day 11: Noise Removal & Morphological Operations

Outcomes:

- Understand what **noise** is in images and masks
- Clean binary masks using **morphological operations**
- Use **erosion, dilation, opening, and closing**
- Improve color detection results

Noise

What is noise?

Noise is any **unwanted pixel** in an image or mask.

In color detection masks:

- Small white dots appear
- Unwanted regions get detected
- Edges look rough

This is called **noise**.

Noise usually comes from:

- Lighting changes
- Shadows
- Camera quality

How is Noise a Problem

A **binary mask** should ideally have:

- White (1) → only the object
- Black (0) → background

Noise breaks this by adding extra white/black pixels, which:

- Confuses object detection
- Reduces accuracy
- Makes contours unreliable

How to Remove Noise

To clean noise, **morphological operations** are applied on the binary mask.

A small matrix called a **kernel** is used to modify pixel shapes.

- **Erosion** removes extra white pixels and small noise.
- **Dilation** adds white pixels to fill gaps in the detected object.
- **Opening** (Erosion → Dilation) removes small noise while preserving object shape.
- **Closing** (Dilation → Erosion) fills holes and connects broken regions.

Using these operations makes the mask cleaner, improves object boundaries, and gives better color detection results.

Morphological Operations

Morphological Operations are **image operations** that work on **binary images** (like masks).

They use a **kernel** (structuring element).

Kernel

A **kernel** (also called structuring element) is a small matrix used in **morphological operations** to decide how pixels are processed during erosion, dilation, opening, and closing.

It defines the **area of influence** around each pixel.

Syntax:

```
kernel = np.ones((rows, cols), dtype)
```

Example:

```
kernel = np.ones((5,5), np.uint8) #uint = unsigned int 8-bit
```

What it does:

- creates a **5×5 matrix filled with 1s**
- sets the data type to **unsigned 8-bit integer** (0–255), which OpenCV expects

The kernel's core function is to:

- Examine neighboring pixels
- Decide whether a pixel should **remain, be removed, or be added**
- Control **how aggressive** the morphological operation is

Larger kernel → stronger effect
Smaller kernel → gentler effect

Erosion

```
eroded = cv.erode(mask, kernel, iterations = 1)
```

Removes white pixels from edges and removes small white noise.

Used when: Mask has small white dots in background

Dilation

```
dilated = cv.dilate(mask, kernel, iterations = 1)
```

Adds white pixels to edges, expands white regions and fills small holes. This makes the object thicker.

Used when: Object looks broken or incomplete

Opening

Opening = Erosion → Dilation

```
opening = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
```

Best for **removing noise**.

- Removes small white spots
- Keeps object shape mostly intact

Used when: Filling small holes inside detected objects

Closing

Closing = Dilation → Erosion

```
closing = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
```

Best for **filling gaps**.

- Fills holes inside object
- Connects broken parts

Used when: Object has black holes or gaps

When and What to Use

Look at your mask and ask:

Mask problem	**Operation**
Small white dots	Erosion
Object too thin	Dilation
Object too noisy	Opening
Holes inside object	Closing

Where to Add Morphological Operations

Recall that,

Color Detection = Convert → Threshold → Mask → Apply

Updated pipeline including Morphological Operations:

Color Detection = Convert → Threshold → Mask → Morphology → Apply

Key-points

- Morphology works on **binary images** (mask).
- Kernel size controls strength
- Opening is usually the first choice

16-erosion.py

```python
import cv2 as cv

import numpy as np


img = cv.imread("OpenCV-codes/images/image4.png")


hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)


cv.namedWindow("Trackbars")


def nothing(x):

    pass


cv.createTrackbar("LH", "Trackbars", 0, 179, nothing)

cv.createTrackbar("LS", "Trackbars", 0, 255, nothing)

cv.createTrackbar("LV", "Trackbars", 0, 255, nothing)

cv.createTrackbar("UH", "Trackbars", 179, 179, nothing)

cv.createTrackbar("US", "Trackbars", 255, 255, nothing)

cv.createTrackbar("UV", "Trackbars", 255, 255, nothing)


while True:


    lh = cv.getTrackbarPos("LH", "Trackbars")

    ls = cv.getTrackbarPos("LS", "Trackbars")

    lv = cv.getTrackbarPos("LV", "Trackbars")

    uh = cv.getTrackbarPos("UH", "Trackbars")

    us = cv.getTrackbarPos("US", "Trackbars")

    uv = cv.getTrackbarPos("UV", "Trackbars")
    lower_bound = np.array([lh, ls, lv])
    upper_bound = np.array([uh, us, uv])

    mask = cv.inRange(hsv, lower_bound, upper_bound)

    # Kernel
    kernel = np.ones((5, 5), np.uint8)

    # Erosion
    eroded = cv.erode(mask, kernel, iterations = 1)

    result = cv.bitwise_and(img, img, mask=eroded)

    cv.imshow("Mask", eroded)
    cv.imshow("Result", result)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cv.destroyAllWindows()
```

17-dilation.py

```python
import cv2 as cv

import numpy as np


img = cv.imread("OpenCV-codes/images/image4.png")


hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)


cv.namedWindow("Trackbars")


def nothing(x):

    pass


cv.createTrackbar("LH", "Trackbars", 0, 179, nothing)

cv.createTrackbar("LS", "Trackbars", 0, 255, nothing)

cv.createTrackbar("LV", "Trackbars", 0, 255, nothing)

cv.createTrackbar("UH", "Trackbars", 179, 179, nothing)

cv.createTrackbar("US", "Trackbars", 255, 255, nothing)

cv.createTrackbar("UV", "Trackbars", 255, 255, nothing)


while True:


    lh = cv.getTrackbarPos("LH", "Trackbars")

    ls = cv.getTrackbarPos("LS", "Trackbars")

    lv = cv.getTrackbarPos("LV", "Trackbars")

    uh = cv.getTrackbarPos("UH", "Trackbars")

    us = cv.getTrackbarPos("US", "Trackbars")

    uv = cv.getTrackbarPos("UV", "Trackbars")
    lower_bound = np.array([lh, ls, lv])
    upper_bound = np.array([uh, us, uv])

    mask = cv.inRange(hsv, lower_bound, upper_bound)

    kernel = np.ones((5, 5), np.uint8)

    # Dilation
    dilated = cv.dilate(mask, kernel, iterations = 1)

    result = cv.bitwise_and(img, img, mask=dilated)

    cv.imshow("Mask", dilated)
    cv.imshow("Result", result)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cv.destroyAllWindows()
```

18-opening.py

```python
import cv2 as cv

import numpy as np


img1 = cv.imread("OpenCV-codes/images/image4.png")

img = cv.resize(img1, (480, 480))


hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)


cv.namedWindow("Trackbars")


def nothing(x):

    pass


cv.createTrackbar("LH", "Trackbars", 0, 179, nothing)

cv.createTrackbar("LS", "Trackbars", 0, 255, nothing)

cv.createTrackbar("LV", "Trackbars", 0, 255, nothing)

cv.createTrackbar("UH", "Trackbars", 179, 179, nothing)

cv.createTrackbar("US", "Trackbars", 255, 255, nothing)

cv.createTrackbar("UV", "Trackbars", 255, 255, nothing)


while True:


    lh = cv.getTrackbarPos("LH", "Trackbars")

    ls = cv.getTrackbarPos("LS", "Trackbars")

    lv = cv.getTrackbarPos("LV", "Trackbars")

    uh = cv.getTrackbarPos("UH", "Trackbars")

    us = cv.getTrackbarPos("US", "Trackbars")
    uv = cv.getTrackbarPos("UV", "Trackbars")

    lower_bound = np.array([lh, ls, lv])
    upper_bound = np.array([uh, us, uv])

    mask = cv.inRange(hsv, lower_bound, upper_bound)

    kernel = np.ones((5, 5), np.uint8)

    # Opening
    opened = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)

    result = cv.bitwise_and(img, img, mask=opened)

    cv.imshow("Mask", opened)
    cv.imshow("Result", result)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cv.destroyAllWindows()
```

19-closing.py

```python
import cv2 as cv

import numpy as np


img1 = cv.imread("OpenCV-codes/images/image4.png")

img = cv.resize(img1, (480, 480))


hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)


cv.namedWindow("Trackbars")


def nothing(x):

    pass


cv.createTrackbar("LH", "Trackbars", 0, 179, nothing)

cv.createTrackbar("LS", "Trackbars", 0, 255, nothing)

cv.createTrackbar("LV", "Trackbars", 0, 255, nothing)

cv.createTrackbar("UH", "Trackbars", 179, 179, nothing)

cv.createTrackbar("US", "Trackbars", 255, 255, nothing)

cv.createTrackbar("UV", "Trackbars", 255, 255, nothing)


while True:


    lh = cv.getTrackbarPos("LH", "Trackbars")

    ls = cv.getTrackbarPos("LS", "Trackbars")

    lv = cv.getTrackbarPos("LV", "Trackbars")

    uh = cv.getTrackbarPos("UH", "Trackbars")

    us = cv.getTrackbarPos("US", "Trackbars")
    uv = cv.getTrackbarPos("UV", "Trackbars")

    lower_bound = np.array([lh, ls, lv])
    upper_bound = np.array([uh, us, uv])

    mask = cv.inRange(hsv, lower_bound, upper_bound)

    kernel = np.ones((5, 5), np.uint8)

    # Closing
    closed = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)

    result = cv.bitwise_and(img, img, mask=closed)

    cv.imshow("Mask", closed)
    cv.imshow("Result", result)

    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cv.destroyAllWindows()
```