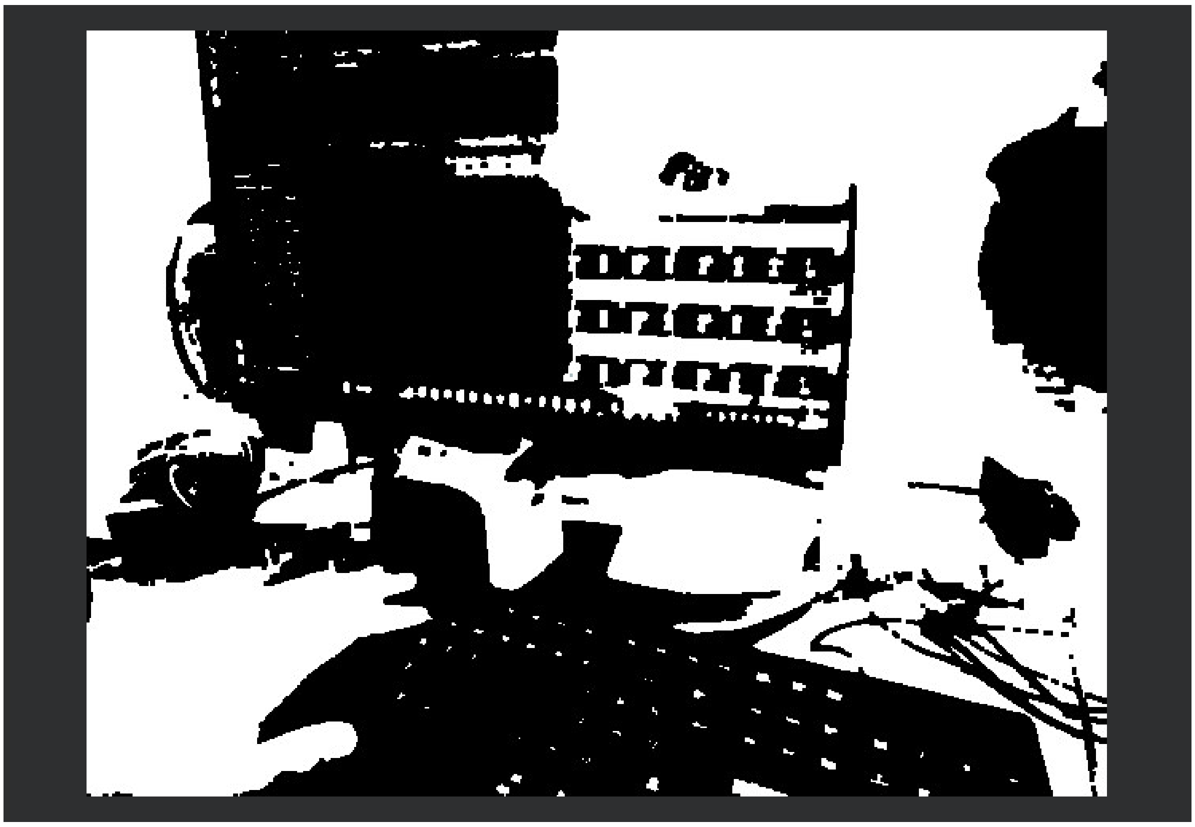
Image Closing

Example Results

Run this section's example code [Source Code/06.cv_lite/14.rgb888_close.py]

In this section, we'll use the `cv_lite` extension module to implement image closing in the RGB888 format on an embedded device.



Principle Explanation

What is Image Closing?

Image closing is a morphological image processing operation used to remove small black holes (noise) or disconnected regions in an image while preserving the overall shape. It is a combination of dilation and erosion operations, with dilation first and erosion second.

- **Dilation**: This operation fills small holes, bridges disconnected areas, and enhances object boundaries by expanding white pixels (foreground) in the image. Mathematically, it scans the image using a convolution kernel (usually rectangular or cross-shaped). If there is at least one white pixel within the kernel, the center pixel is set to white.
- **Erosion**: This operation removes noise or fine structure at the edges of an image by shrinking white areas. It requires that all pixels within the kernel are white before setting the center pixel to white, thereby "eroding" away small, isolated white areas.
- **Effect of the Closing Operation**: Closing = Dilation (original image) + Erosion (dilation result). This removes fine black noise while connecting adjacent white areas. Closing is often used to process binary images or enhance object connectivity during preprocessing, such as removing noise in object detection or filling holes in medical images.

In practical applications, closing parameters (such as kernel size and number of iterations) affect the intensity of the processing, while the binarization threshold determines how the image is binarized (if necessary).

Code Overview

Importing Modules

```
import time, os, gc
from machine import Pin
from media.sensor import * # Camera interface
from media.display import * # Display interface
from media.media import * # Media manager
import cv_lite # AI CV extension (close function)
import ulab.numpy as np # NumPy-like ndarray for MicroPython
```

Setting the Image Size

```
image_shape = [480, 640] # Height x Width
```

Define the image resolution to 480x640 (height x width). The camera and display modules will be initialized based on this size.

Initialize the camera (RGB888 format)

```
sensor = Sensor(width=1280, height=960)
sensor.reset()
sensor.set_framesize(width=image_shape[1], height=image_shape[0])
sensor.set_pixformat(Sensor.RGB888) # Set pixel format to RGB888
```

- Initialize the camera and set the initial resolution to 1280x960 (the frame rate is not specified, so the default value is used).
- Resize the output frame to 640x480 and set it to RGB888 format (three-channel color image, suitable for morphological operations on color images).

Initialize the display module

```
Display.init(Display.ST7701, width=image_shape[1], height=image_shape[0],
to_ide=True)
```

Initialize the display module, using the ST7701 driver chip, with a resolution consistent with the image size. `to_ide=True` indicates that the image will be simultaneously transferred to the IDE for virtual display.

Initialize the media manager and start the camera

```
MediaManager.init()
sensor.run()
```

Initialize the media resource manager and start the camera to capture the image stream.

Set closing parameters

```
kernel_size = 3 # Kernel size (odd recommended)
iterations = 1 # Number of morphological passes
threshold_value = 100 # Binarization threshold (0 = Otsu)
clock = time.clock() # Start FPS timer
```

- `kernel_size`: Kernel size, used for morphological operations (dilation and erosion). An odd number is recommended; larger values increase the processing range.
- `iterations`: Number of closing iterations, indicating the number of times dilation and erosion are repeated. A larger value increases the effect.
- `threshold_value`: Binarization threshold, used before or after the closing operation. A value of 0 uses the Otsu thresholding algorithm.
- `clock`: used to calculate frame rate.

Image processing and closing operation

```
while True:
    clock.tick()  # Start timing / Start frame timing

    # Capture a frame / Capture a frame
    img = sensor.snapshot()
    img_np = img.to_numpy_ref() # Get RGB888 ndarray reference / Get RGB888
ndarray reference (HWC)

    # Apply closing operation (dilate then erode) / Apply closing operation
(dilate then erode) / result_np = cv_lite.rgb888_close(
    result_np = cv_lite.rgb888_close(
        image_shape,
        img_np,
        kernel_size,
        iterations,
        threshold_value
    )

    # Construct an image object (grayscale image) for display / Wrap processed
grayscale image for display
    img_out = image.Image(image_shape[1], image_shape[0], image.GRAYSCALE,
                            alloc=image.ALLOC_REF, data=result_np)

    # Show closed image

    Display.show_image(img_out)

    # Clean up memory and print FPS

    gc.collect()
    print("close fps:", clock.fps())
```

- **Image capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Closure processing**: Call `cv_lite.rgb888_close()` to perform the closing operation, returning the processed image as a NumPy array.
- **Image packaging and display**: Package the processed result into a grayscale image object and display it on the screen or in an IDE virtual window.
- **Memory management and frame rate output**: Call `gc.collect()` to clean up memory and print the current frame rate using `clock.fps()`.

Resource release

```
sensor.stop() # Stop sensor
Display.deinit() # Deinitialize display
os.exitpoint(os.EXITPOINT_ENABLE_SLEEP) # Set safe exit point
time.sleep_ms(100) # Delay
MediaManager.deinit() # Release media resources
```

Parameter adjustment instructions

- `kernel_size`: Convolution kernel size. A larger value results in a larger processing area, which is suitable for removing larger noises. It is recommended to use an odd number (such as 3, 5, or 7). Start with 3 to balance detail preservation and noise removal.
- `iterations`: Number of iterations. A larger value results in a stronger closing operation, but it increases the amount of computation and may over-smooth the image. It is recommended to start with 1 and gradually increase it based on the image noise level.
- `threshold_value`: Binarization threshold. Higher values require brighter pixels to be retained. A value of 0 uses Otsu's automatic threshold. It is recommended to test from 50 to 150 based on the image brightness, or set to 0 for automatic adaptation.