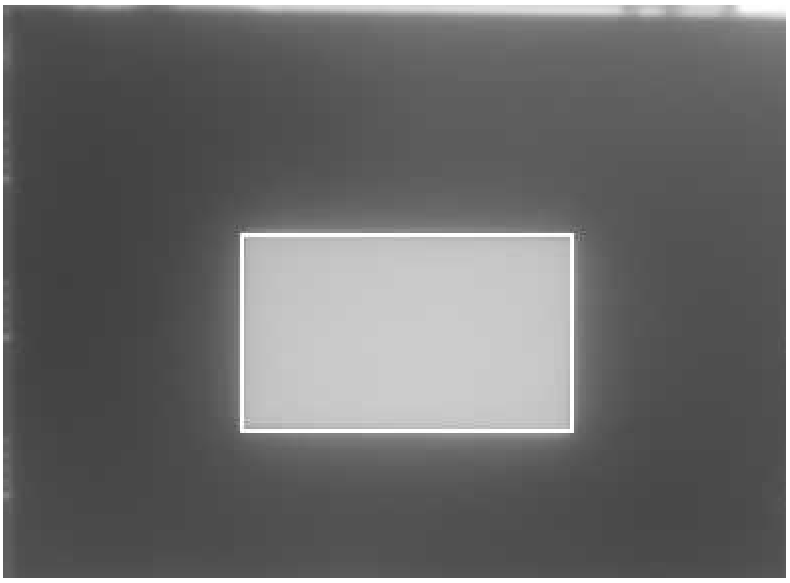
Rectangle Recognition - Grayscale Image

Example Results

Run this section's example code [Source Code/06.cv_lite/9.grayscale_find_rectangle.py]

In this section, we will use the `cv_lite` extension module to implement grayscale image rectangle detection on an embedded device.



Code Overview

Importing Modules

```python
import time, os, sys, gc
from machine import Pin
from media.sensor import * # Camera interface
from media.display import * # Display interface
from media.media import * # Media manager
import _thread
import cv_lite # cv_lite extension (C bindings)
import ulab.numpy as np # MicroPython NumPy library
```

Setting the Image Size

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

Define the image resolution to 480x640 (Height x Width). width), the camera and display module will be initialized based on this size later.

Initialize the camera (grayscale mode)

```
sensor = Sensor(id=2, width=1280, height=960, fps=90)
sensor.reset()
sensor.set_framesize(width=image_shape[1], height=image_shape[0])
sensor.set_pixformat(Sensor.GRAYSCALE) # Grayscale format
```

- Initialize the camera, set the resolution to 1280x960 and the frame rate to 90 FPS.
- Resize the output frame to 640x480 and set it to grayscale mode (single-channel image, saves memory and computing resources, suitable for rectangle detection).

Initialize the display module

```
Display.init(Display.ST7701, width=image_shape[1], height=image_shape[0],
             to_ide=True, quality=50)
```

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 transmitted to the IDE for virtual display, and `quality=50` sets the image transmission quality.

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.

Optional gain setting (brightness/contrast adjustment)

```
gain = k_sensor_gain()
gain.gain[0] = 20
sensor.again(gain)
```

Set the camera's gain parameters to adjust brightness and contrast to optimize image quality. `gain[0] = 20` is used to increase image brightness.

Setting rectangle detection parameters

```
canny_thresh1 = 50 # Canny low threshold
canny_thresh2 = 150 # Canny high threshold
approx_epsilon = 0.04 # Polygon approximation accuracy (smaller, more accurate)
area_min_ratio = 0.001 # Minimum area ratio (relative to the total image area)
max_angle_cos = 0.3 # Maximum angle cosine between edges (smaller, closer to a
rectangle)
gaussian_blur_size = 5 # Gaussian blur kernel size (odd number)
clock = time.clock() # Start FPS timer
```

- `canny_thresh1`: Canny The lower threshold for edge detection, used to control the detection of weak edges.
- `canny_thresh2`: The upper threshold for Canny edge detection, used to control the detection of strong edges. A larger value results in fewer edges being detected.
- `approx_epsilon`: The polygon fitting accuracy ratio. A smaller value results in more accurate fitting, but the computational effort increases.
- `area_min_ratio`: The minimum area ratio relative to the total image area, used to filter out rectangles that are too small.
- `max_angle_cos`: The maximum angle cosine. A smaller value results in the detected shape being closer to a rectangle (used to determine if it is a rectangle).
- `gaussian_blur_size`: The Gaussian blur kernel size. Used for image preprocessing to reduce noise. Must be an odd number.
- `clock`: Used to calculate the frame rate.

Image Processing and Rectangle Detection

```
while True:
    clock.tick()

    # Capture a frame
    img = sensor.snapshot()
    img_np = img.to_numpy_ref()

    # Call the underlying rectangle detection function
    # Return format: [x0, y0, w0, h0, x1, y1, w1, h1, ...]
    rects = cv_lite.grayscale_find_rectangles(
        image_shape, img_np,
        canny_thresh1, canny_thresh2,
        approx_epsilon,
        area_min_ratio,
        max_angle_cos,
        gaussian_blur_size
    )
    # Iterate over the detected rectangles and draw
    for i in range(0, len(rects), 4):
        x = rects[i]
        y = rects[i + 1]
        w = rects[i + 2]
        h = rects[i + 3]
        img.draw_rectangle(x, y, w, h, color=(255, 255, 255), thickness=2)

    # Display image
    Display.show_image(img)

    # Garbage collection and print FPS and rectangle count
    gc.collect()
    print("fps:", clock.fps(), "rects:", len(rects) // 4)
```

- **Image capture**: Get an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Rectangle Detection**: Call `cv_lite.grayscale_find_rectangles()` to detect rectangles and return a list of detected rectangle information (each rectangle occupies 4 elements: `[x, y, w, h]`, i.e., the coordinates of the upper-left corner, width, and height).

- **Rectangle Drawing**: Iterate over each detected rectangle and draw a white rectangular outline (`color=(255, 255, 255)`) on the image to mark the target area.
- **Image Display**: Display the processed image to the screen or IDE virtual window.
- **Memory Management and Frame Rate Output**: Call `gc.collect()` to clear memory and use `clock.fps()` to print the current frame rate and the number of detected rectangles.

Resource release

```
sensor.stop()
Display.deinit()
os.exitpoint(os.EXITPOINT_ENABLE_SLEEP)
time.sleep_ms(100)
MediaManager.deinit()
```

Parameter adjustment instructions

- `canny_thresh1`: Low threshold. The smaller the value, the more weak edges can be detected, but it may introduce noise. It is recommended to start adjusting from 50.
- `canny_thresh2`: High threshold. The larger the value, the fewer strong edges are detected. It is suitable for noisy environments. Increasing this value can reduce false edges.
- `approx_epsilon`: Polygon fitting accuracy ratio. The smaller the value, the more accurate the fitting, but the calculation amount increases. It is recommended to adjust it according to the complexity of the image.
- `area_min_ratio`: Minimum area ratio. The larger the value, the more rectangles are filtered out. It is suitable for ignoring small targets. Adjust it according to the target size.
- `max_angle_cos`: Maximum angle cosine. Smaller values result in detected shapes that are closer to rectangles, making it suitable for strictly filtering rectangles. A starting value of 0.3 is recommended.
- `gaussian_blur_size`: Gaussian blur kernel size. Larger values result in greater image smoothing, but may result in loss of detail. This must be an odd number; 3 or 5 is recommended as a starting point for testing.