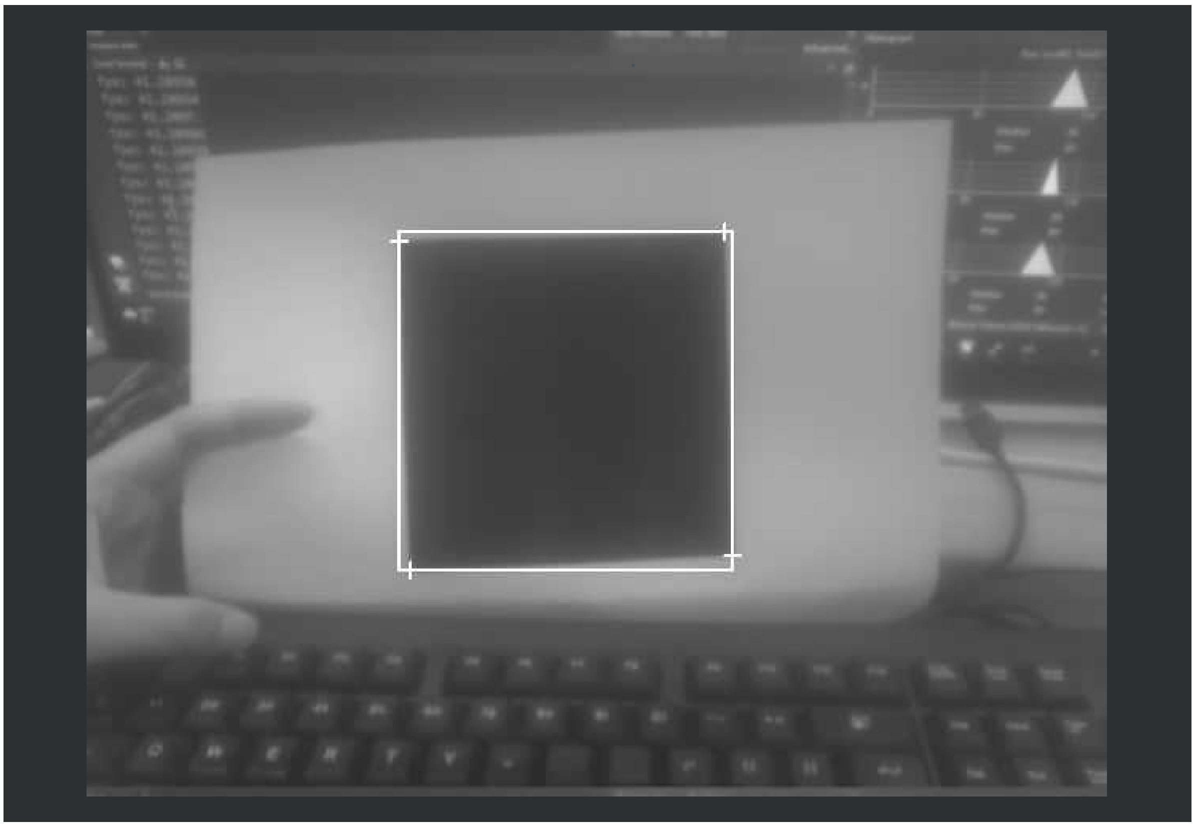
Rectangle recognition combined with corner recognition - grayscale image

Example Results

Run this section's example code [Source Code/06.cv_lite/5.grayscale_find_rectangles_with_corners.py]

In this section, we'll use the `cv_lite` extension module to implement rectangle detection in grayscale images on an embedded device, drawing rectangles and corner points.



Principle Explanation

What is Grayscale Image Rectangle Detection?

Grayscale image rectangle detection is a computer vision technique used to identify and locate rectangular objects in grayscale images. Compared to color images, grayscale images contain only brightness information, which simplifies the processing and reduces computational effort. The detection process is typically based on edge detection and contour analysis, using a series of image processing steps (such as blurring, edge extraction, and contour fitting) to find regions that meet rectangular features. This code uses the `cv_lite` module to implement rectangle detection in grayscale images and returns the position, size, and coordinates of the rectangle's four corner points.

- **How it works**: Grayscale image rectangle detection includes the following steps:
    1. **Preprocessing**: Gaussian blur is applied to the grayscale image to reduce noise.
    2. **Edge Detection**: The Canny edge detection algorithm is used to extract edges in the image, and strong and weak edges are determined based on two thresholds (low and high).
    3. **Contour Extraction**: Contours are extracted from the edge image and polygon-fitted to the contours to identify shapes that approximate rectangles.
    4. **Rectangle Filtering**: Contours that meet rectangular characteristics are further filtered based on criteria such as area ratio and angle cosine, and the bounding box and corner coordinates of the rectangle are calculated.
    5. **Result**: The position (x, y), size (width, height), and coordinates of the rectangle's four corner points are returned.
- **Effect**: In grayscale images, rectangle detection accurately identifies rectangular objects and draws a rectangular box and corner points on the image. Because grayscale images ignore color information, they process faster but may be more sensitive to lighting changes. Detection results are affected by parameters such as the Canny threshold, area ratio, and angle cosine, and require adjustment based on the actual image characteristics.
- **Application Scenarios**: Grayscale image rectangle detection is widely used in scenarios requiring fast processing, such as object detection (e.g., identifying cards and signs), robot vision (e.g., locating objects), augmented reality (AR), and document processing (e.g., detecting table or page boundaries). Using grayscale images reduces computing resource requirements and is suitable for embedded devices.

In rectangle detection, the choice of parameters significantly impacts detection accuracy and performance. For example, the Canny threshold determines the sensitivity of edge detection, the area ratio filters out rectangles that are too small, and the angle cosine controls the regularity of the rectangles.

Code Overview

Importing Modules

```
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). The camera and display modules will be initialized based on this size.

Initialize the camera (grayscale format)

```
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 format (single-channel image, suitable for fast processing and edge 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.

Set camera gain (brightness/contrast adjustment)

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

Set the camera gain value (`gain[0] = 20`) to adjust the brightness and contrast of the image to optimize the grayscale image quality and facilitate edge detection.

Set the frame rate timer

```
clock = time.clock() # Start FPS timer
```

Initialize the frame rate timer, which is used to calculate and display the number of frames processed per second (FPS).

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)
```

- `canny_thresh1` and `canny_thresh2`: Canny low and high thresholds, used to control the sensitivity of edge detection.
- `approx_epsilon`: Polygon fitting accuracy ratio. A smaller value results in a more accurate fit.
- `area_min_ratio`: Minimum area ratio relative to the total image area, used to filter out rectangles that are too small.
- `max_angle_cos`: Maximum angle cosine value, used to determine if a contour approximates a rectangle.
- `gaussian_blur_size`: Gaussian blur kernel size. Must be an odd number. Used for image preprocessing to reduce noise.

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, c1.x, c1.y, c2.x, c2.y, c3.x, c3.y, c4,x,
c4.y], [x1, y1, w1, h1,c1.x, c1.y, c2.x, c2.y,     c3.x, c3.y, c4,x, c4.y], ...]
    rects = cv_lite.grayscale_find_rectangles_with_corners(
        image_shape, img_np,
        canny_thresh1, canny_thresh2,
        approx_epsilon,
        area_min_ratio,
        max_angle_cos,
        gaussian_blur_size
    )
    # Traverse the detected rectangles and draw the rectangle frame and corner
points
    for i in range(len(rects)):
        r = rects[i]
        img.draw_rectangle(r[0], r[1], r[2], r[3], color=(255, 255, 255),
thickness=2)
```

```
            img.draw_cross(r[4], r[5], color=(255, 255, 255), size=5, thickness=2)
            img.draw_cross(r[6], r[7], color=(255, 255, 255), size=5, thickness=2)
            img.draw_cross(r[8], r[9], color=(255, 255, 255), size=5, thickness=2)
            img.draw_cross(r[10], r[11], color=(255, 255, 255), size=5, thickness=2)

        # Display image / Show image
        Display.show_image(img)

        # Garbage collection & output frame rate / Garbage collect and print FPS
        gc.collect()
        print("fps:", clock.fps())
```

- **Image capture**: Get a frame of grayscale image through `sensor.snapshot()` and convert it to a NumPy array reference through `to_numpy_ref()`.
- **Rectangle detection processing**: Call `cv_lite.grayscale_find_rectangles_with_corners()` function to perform rectangle detection and return a list of detected rectangles. Each rectangle contains the position (x, y), size (width, height) and the coordinates of the four corner points.
- **Drawing Rectangles and Corner Points**: Iterate over the detected rectangle list, draw the rectangle on the image using `draw_rectangle()`, and draw a white cross mark at each corner using `draw_cross()`.
- **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 print the current frame rate using `clock.fps()`.

Resource release

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

Parameter adjustment instructions

- `canny_thresh1` **and** `canny_thresh2`: Canny edge detection thresholds. Low thresholds (30-70 recommended) are used to connect weak edges, while high thresholds (100-200 recommended) are used to identify strong edges. Too low a value may result in excessive noise edges, while too high a value may miss true edges. Adjust according to image contrast.
- `approx_epsilon`: Polygon fitting accuracy ratio. Smaller values (such as 0.01-0.05) result in more accurate fitting, but the computational effort increases. It is recommended to start testing with 0.02.
- `area_min_ratio`: Minimum area ratio. Larger values (e.g., 0.01) filter out more small rectangles, while smaller values (e.g., 0.001) retain more small rectangles. We recommend adjusting this value based on the target size.
- `max_angle_cos`: Maximum angle cosine. Smaller values (e.g., 0.1-0.3) require the outline to be closer to a rectangle (with right angles), while larger values allow for more distortion. We recommend starting with 0.2.
- `gaussian_blur_size`: Gaussian blur kernel size. Must be an odd number. Larger values (e.g., 5-9) result in stronger denoising, but may lose detail. We recommend starting with 3 or 5.