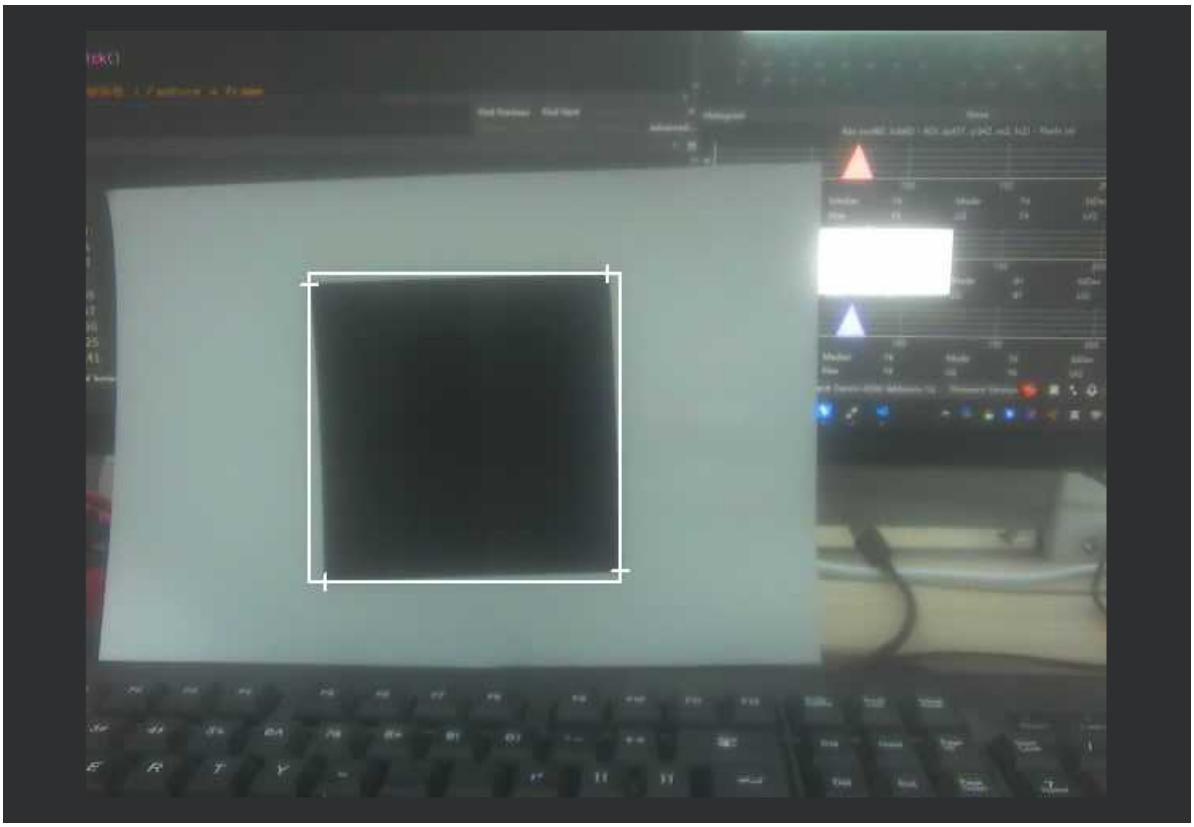
Rectangle recognition combined with corner recognition - color image

Example Results

Run this section's example code [Source Code/06.cv_lite/6.rgb888_find_rectangle_with_corners.py]

In this section, we'll use the `cv_lite` extension module to implement rectangle detection for RGB888 images on an embedded device and draw a rectangle and its corner points.



Principle Explanation

What is Image Rectangle Detection?

Image rectangle detection is a computer vision technique used to identify and locate rectangular objects in an image. Rectangle detection 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 and return the position, size, and coordinates of the rectangle's four corner points.

- **Operation Principle**: Rectangle detection generally involves the following steps:
    1. **Preprocessing**: Apply a Gaussian blur to the image to reduce noise.
    2. **Edge Detection**: Use the Canny edge detection algorithm to extract edges in the image, determining strong and weak edges based on two thresholds (low and high).
    3. **Contour Extraction**: Extract contours from the edge image and perform polygon fitting on them to identify shapes that approximate rectangles.
    4. **Rectangle Filtering**: Further filter contours that meet rectangular characteristics based on criteria such as area ratio and angle cosine, and calculate the bounding box and corner coordinates of the rectangle.
    5. **Result Return**: Returns the rectangle's position (x, y), size (width, height), and coordinates of its four corner points.
- **Effect**: Rectangle detection accurately identifies rectangular objects in an image and draws a rectangular bounding box and corner points on the image. It is suitable for scenarios where regular shapes need to be located. Detection results are affected by parameters such as the Canny threshold, area ratio, and angle cosine, and should be adjusted based on the actual image characteristics.
- **Application Scenarios**: Rectangle detection is widely used in object detection (such as identifying cards and signs), robot vision (such as locating objects), augmented reality (AR), and document processing (such as detecting table or page boundaries).

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 rectangle.

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 (RGB888 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.RGB888) # RGB888 format / rgb888 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 RGB888 format (three-channel color image, suitable for rectangle detection in color images).

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 image brightness and contrast to optimize 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. A low threshold is used to connect weak edges, while a high threshold is used to identify strong edges.
- `approx_epsilon`: The polygon fitting accuracy ratio. A smaller value results in a more accurate fit, 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 value, used to determine whether a contour approximates a rectangle. A smaller value requires a closer approximation to a right angle.
- `gaussian_blur_size`: The 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.rgb888_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 image through `sensor.snapshot()` and convert it to a NumPy array reference through `to_numpy_ref()`.
- **Rectangle detection processing**: Call `cv_lite.rgb888_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.