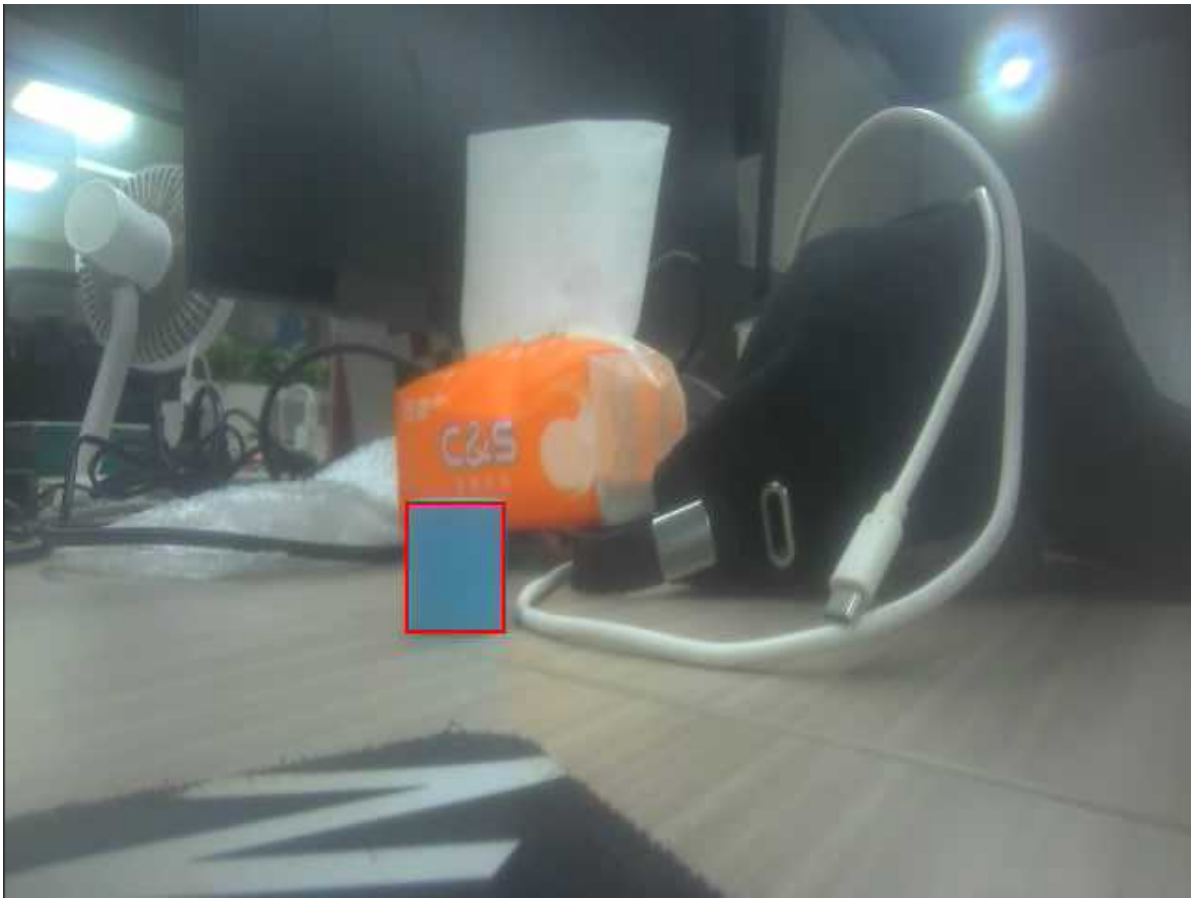
Color Block Recognition - Color Image

Example Results

Run this section's example code [Source Code/06.cv_lite/2.rgb888_find_blobs.py]

This section is based on color block recognition for RGB888 (color) images. The K230 will detect blocks in the image that meet the set threshold and mark them with a rectangular box.

> Observe the frame rate output on the serial terminal. When the power supply is stable, the frame rate can be stable at around 55 frames.



Code Explanation

Importing Modules

```
import time, os, sys, gc
from machine import Pin
from media.sensor import * # Import the camera interface
from media.display import * # Import the display interface
from media.media import * # Import the media manager
import _thread
import cv_lite # cv_lite extension module
import ulab.numpy as np # MicroPython NumPy library
```

These modules provide the necessary functionality for camera control, image display, memory management, and color block detection.

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 mode)

```
sensor = Sensor(id=2, 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 and set the frame rate to 90 FPS.
- Use the default camera resolution.
- Resize the output frame to 640x480 and set it to RGB888 format (three-channel color image, suitable for color 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 LCD 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 color blob detection parameters

```
threshold = [120, 255, 0, 50, 0, 50] # Format: [Rmin, Rmax, Gmin, Gmax, Bmin, Bmax]
min_area = 100 # Minimum blob area
kernel_size = 1 # Kernel size for morphological ops (used for preprocessing)
clock = time.clock() # Start FPS timer
```

- `threshold`: Color detection threshold range, in the format `[Rmin, Rmax, Gmin, Gmax, Bmin, Bmax]`, representing the value range of the three RGB channels. Here, detection is set to red (R channel range is 120-255, G and B channels range is 0-50).
- `min_area`: The minimum area of a connected component. Areas smaller than this value are ignored.
- `kernel_size`: The kernel size for image erosion and dilation operations, used for preprocessing images to remove noise.
- `clock`: Used for calculating frame rate.

Image Processing and Color Block Detection

```
while True:
    clock.tick()

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

    # Call the cv_lite extension for color block detection, returning a list of [x, y, w, h, ...]
    blobs = cv_lite.rgb888_find_blobs(image_shape, img_np, threshold, min_area, kernel_size)

    # Iterate over the detected color blocks and draw rectangular boxes
    for i in range(len(blobs) // 4): # Correct for integer division
        x = blobs[4*i]
        y = blobs[4*i + 1]
        w = blobs[4*i + 2]
        h = blobs[4*i + 3]
        img.draw_rectangle(x, y, w, h, color=(255, 0, 0), thickness=2) # Red frame

    # Display the resulting image / Show image with blobs
    Display.show_image(img)

    # Print FPS
    print("finddblobs:", clock.fps())

    # Garbage collection
    gc.collect()
```

- **Image capture**: Use `sensor.snapshot()` to obtain a frame of image and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Color block detection**: Call `cv_lite.rgb888_find_blobs()` to detect color blocks and return a list of detected area information (each blob occupies 4 elements: `[x, y, w, h]`, i.e., the coordinates of the upper left corner and the width and height).

- **Draw rectangle**: Traverse each detected blob and draw a red rectangle (`color=(255, 0, 0)`) on the image to mark the target area.
- **Display image**: Display the processed image to the screen or IDE virtual window.
- **Memory management and frame rate output**: Call `gc.collect()` to clean up the memory and print the current frame rate through `clock.fps()`.

Resource release

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

When the program exits, stop the camera, release the display module resources, and clean up the media manager.

Parameter adjustment instructions

- `threshold`: Adjust the threshold range of the RGB channel according to the target color. For example:
  - Detecting red: `[120, 255, 0, 50, 0, 50]` (R high, G and B low).
  - Detecting green: `[0, 50, 120, 255, 0, 50]` (G high, R and B low).
  - Detecting blue: `[0, 50, 0, 50, 120, 255]` (B high, R and G low).
- `min_area`: Adjust this value based on the object size. Smaller values detect more small areas, while larger values filter out noise.
- `kernel_size`: Used for image erosion and dilation. Larger values produce stronger noise reduction, but may lose detail in small objects.