Color Block Recognition - Grayscale Image

Example Results

Run this section's example code [Source Code/06.cv_lite/1.grayscale_find_blobs.py]

This section is based on color block recognition in grayscale images. The K230 will detect color blocks in the image that meet the set threshold and mark them with rectangular boxes.

> Observe the frame rate information output by the serial terminal. When the power supply is stable, the frame rate can be stable at over 80 frames.

Code Explanation

Code Snippet Explanation

Importing Modules

```
import time, os, sys, gc
from machine import Pin
from media.sensor import * # Camera module
from media.display import * # Display module
from media.media import * # Media manager
import _thread
import cv_lite # cv_lite extension module
import ulab.numpy as np # Lightweight NumPy
```

These modules provide the functionality required for camera control, image display, memory management, and blob 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

```
sensor = Sensor(id=2, width=1280, height=720, fps=90) # Construct the Sensor
object
sensor.reset() # Reset the camera
sensor.set_framesize(width=image_shape[1], height=image_shape[0]) # Set the
frame size
sensor.set_pixformat(Sensor.GRAYSCALE) # Set to grayscale output
```

- Initialize the camera, set the resolution to 1280x720 and the frame rate to 90 FPS.
- Resize the output frame to 640x480 and set it to grayscale mode (single-channel image, saving memory and computing resources).

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 (i.e., the LCD screen of the K230 Deluxe Edition), with the 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.

3.5 Initialize the media manager and start the camera

```
MediaManager.init()
sensor.run() # Start the camera image stream
```

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

Set connected component detection parameters

```
threshold = [230, 255] # Binarization threshold range (bright area)
min_area = 10 # Minimum target area
kernel_size = 1 # Erosion kernel size (for noise reduction)
clock = time.clock() # Frame rate timer
```

- `threshold`: Binarization threshold range. Pixels with grayscale values between 230 and 255 are considered target areas (suitable for detecting bright areas).
- `min_area`: Minimum connected component area. Areas smaller than this value are ignored.
- `kernel_size`: Kernel size for the image erosion operation, used for noise removal.
- `clock`: Used to calculate the frame rate.

Image processing and blob detection

```
while True:
    clock.tick()

    # Get a frame of image
    img = sensor.snapshot()
    img_np = img.to_numpy_ref() # Get image data reference

    # Perform grayscale binary connected area detection
    blob = cv_lite.grayscale_find_blobs(
        image_shape, img_np,
        threshold[0], threshold[1],
        min_area, kernel_size
    )

    # If a connected area is detected
    if len(blob) > 0:
        # blob = [x, y, w, h]
        img.draw_rectangle(blob[0], blob[1], blob[2], blob[3],
                           color=(0, 0, 0), thickness=2) # Draw a black
rectangle

    # Display the image
    Display.show_image(img)

    # Clear memory and print the frame rate.
    gc.collect()
    print("findblobs:", clock.fps())
```

- **Image Capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Blob Detection**: Call `cv_lite.grayscale_find_blobs()` to detect connected components and return the detected region information (in the format `[x, y, w, h]`, i.e., the

coordinates of the upper-left corner and the width and height).

- **Draw Rectangle**: If a blob is detected, draw a black rectangle on the image to mark the target region.
- **Display Image**: 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()
```

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

Parameter adjustment instructions

- `threshold`: Adjust the binarization threshold range according to the ambient light. If the target area is dark, you can reduce the range (such as `[100, 200]`); if the target area is bright, you can increase the range (such as `[200, 255]`).
- `min_area`: Adjust this value according to the target size. Smaller values will detect more small areas, and larger values will filter out noise.
- `kernel_size`: Used for image erosion. A larger value will have a stronger noise reduction effect, but may lose details of small targets.