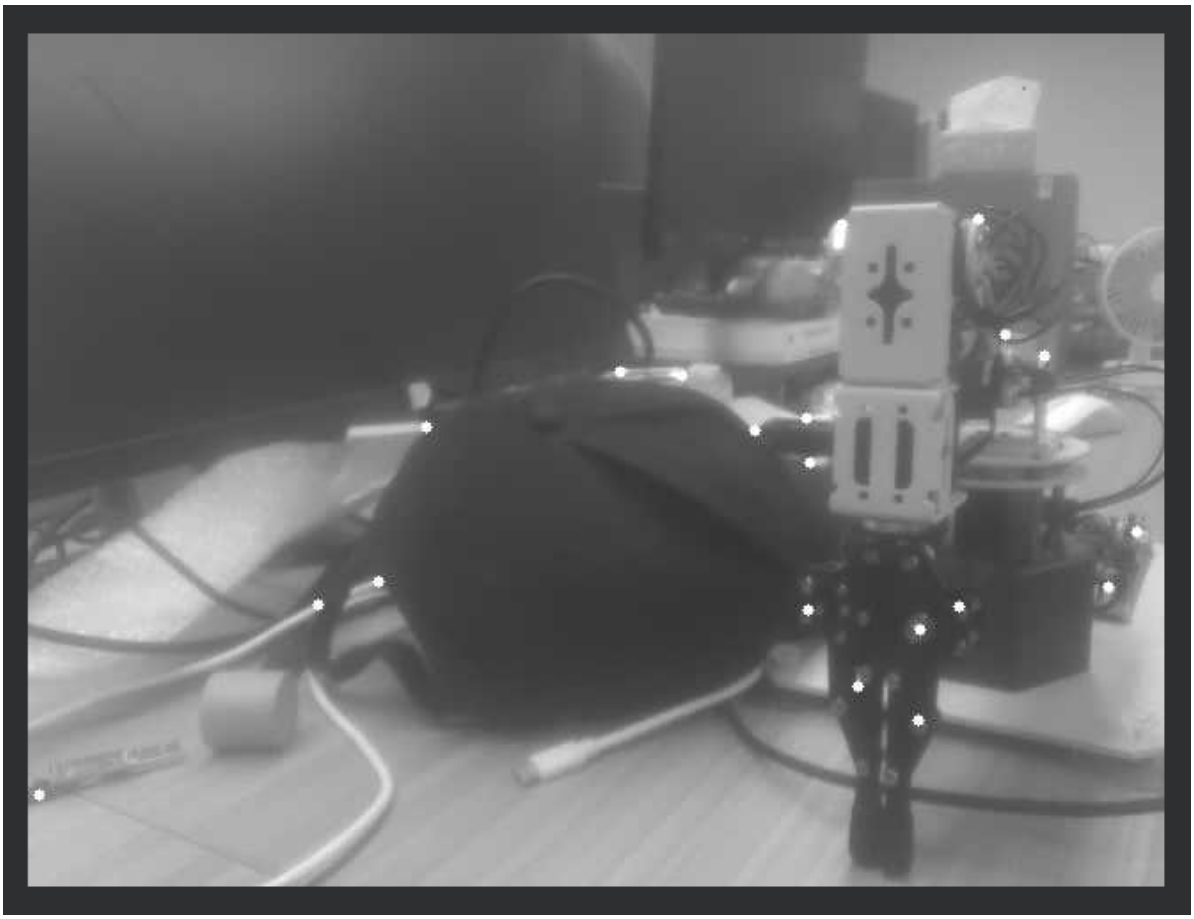
Corner Recognition-Grayscale Image

Example Results

Run this section's example code [Source Code/06.cv_lite/29.grayscale_find_corners]

In this section, we will use the `cv_lite` extension module to implement grayscale image corner detection on an embedded device. We will detect and display key corners in the image for feature extraction and object tracking.



Principle Explanation

What is Grayscale Image Corner Detection?

Grayscale image corner detection is a computer vision technique used to identify points in an image with dramatic brightness changes (i.e., corners). These points are typically edges or corners of objects and are highly distinctive. Corner detection is based on grayscale images. Common algorithms, such as the Shi-Tomasi corner detector, can efficiently extract stable key points.

- **Operation Principle**: Corner detection algorithms (such as Shi-Tomasi) first calculate the image's gradient matrix (Hessian matrix) and then evaluate the corner response value of each pixel (based on the eigenvalues of a local window). High-quality corners are selected by setting a quality threshold and a minimum distance. The formula involves calculating the pixel's eigenvalues λ1 and λ2. If min(λ1, λ2) > the threshold, the pixel is considered a corner. This process is performed on grayscale images to reduce computational effort.
- **Effect**: The detection results highlight key points in the image and can be used for subsequent processing such as object tracking or matching. After processing, corners are marked on the image, improving the accuracy of visual analysis. However, detection accuracy is affected by image noise and parameters.
- **Application Scenarios**: Grayscale image corner detection is widely used in real-time object tracking (such as SLAM), motion detection, image registration, and robotic vision. On embedded devices, its high computational efficiency makes it suitable for resource-constrained environments, such as real-time camera processing.

In practical applications, by adjusting parameters, the robustness and number of detections can be optimized to suit different image scenarios.

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 module
import ulab.numpy as np
```

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=60)
sensor.reset()
sensor.set_framesize(width=image_shape[1], height=image_shape[0])
sensor.set_pixformat(Sensor.GRAYSCALE)
```

- Initialize the camera, set the resolution to 1280x960 and the frame rate to 60 FPS.
- Resize the output frame to 640x480 and set it to GRAYSCALE format (single-channel grayscale image, suitable for corner detection to reduce computational complexity).

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 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.

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 the frame rate timer

```
clock = time.clock() # Start the frame rate timer / Start FPS timer
```

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

Set corner detection parameters

```
max_corners = 20 # Maximum number of corners
quality_level = 0.01 # Corner quality factor (0.01 to 0.1)
min_distance = 20.0 # Minimum distance between corners
```

- `max_corners`: Maximum number of detected corners, limiting the output results.
- `quality_level`: Corner quality threshold, used to filter out corners with high response values.
- `min_distance`: Minimum distance between corners, to avoid detecting too dense points.

Image Processing and Corner Detection

```
while True:
    clock.tick()

    # Capture current frame
    img = sensor.snapshot()
    img_np = img.to_numpy_ref() # Get GRAYSCALE ndarray reference

    # Call corner detection function, return corner array [x0, y0, x1, y1, ...]
    corners = cv_lite.grayscale_find_corners(
        image_shape, img_np,
        max_corners,
        quality_level,
        min_distance
    )
```

```
    # Traverse corner array and draw detected corners
    for i in range(0, len(corners), 2):
        x = corners[i]
        y = corners[i + 1]
        img.draw_circle(x, y, 3, color=(255, 255, 255), fill=True)

    # Display image with corners
    Display.show_image(img)

    # Perform garbage collection
    gc.collect()

    # Print current FPS and number of corners
    print("fps:", clock.fps(), "| corners:", len(corners) // 2)
```

- **Image Capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Corner Detection**: Call `cv_lite.grayscale_find_corners()` to perform corner detection and return an array of corner coordinates.
- **Drawing and Display**: Iterate over the detected corners, draw circle markers on the image, and display them on the screen or in an IDE virtual window.
- **Memory management and frame rate output**: Call `gc.collect()` to clear memory, and print the current frame rate and number of corner points through `clock.fps()`.

Resource release

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

Parameter adjustment instructions

- `max_corners`: Maximum number of corner points. A larger value (e.g., 10-50) detects more corner points, but may increase the computational burden. A smaller value (e.g., 5) reduces the number of points to increase speed. It is recommended to start testing with 20 based on image complexity.
- `quality_level`: Shi-Tomasi quality factor. Higher values (e.g., 0.01-0.1) require stronger corner response, resulting in more accurate detection but potentially fewer corners. A value too low (e.g., 0.001) may detect noise. It is recommended to start fine-tuning from 0.01.
- `min_distance`: Minimum corner distance. Higher values (e.g., 10-50) ensure corner spacing and avoid densely packed corners. Lower values (e.g., 5) result in denser detection but may include redundancy. It is recommended to adjust this value based on image resolution and target scene. A starting value of 20 is recommended for testing.