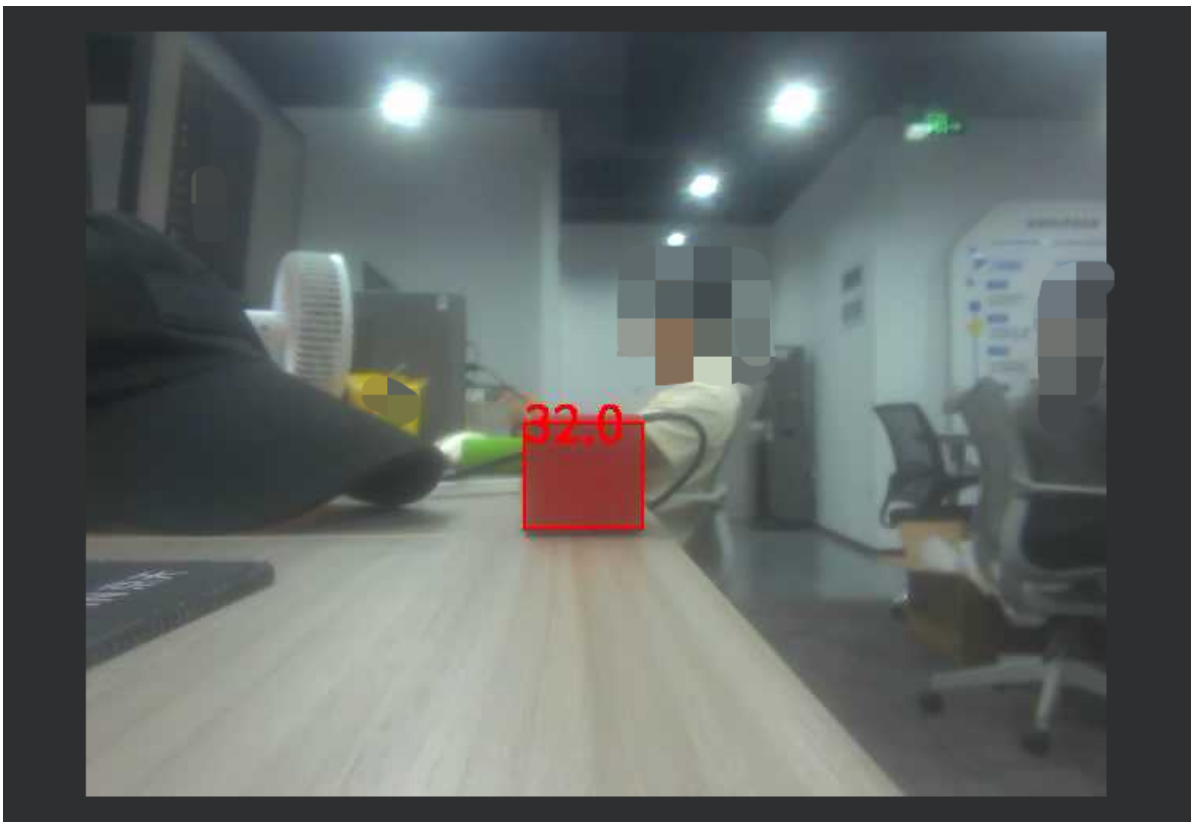
PnP ranging

Example Results

Run this section's example code [Source Code/06.cv_lite/23.rgb888_find_blobs_and_distance.py]

In this section, we'll use the `cv_lite` extension module to implement color blob detection in RGB888 format images on an embedded device and estimate the distance of the detected blob using the PnP algorithm.

> The measured distance is 32.5cm. The code performs integer division on the distance. To improve accuracy, consider changing the integer division (//) to decimal division (/).



Principle Explanation

What is Image Color Blob Detection and Distance Estimation?

Image color blob detection is a computer vision technique used to identify regions of specific color (called blob) in an image and estimate the distance of these blob in the real world through further calculations. Color block detection segments the image based on color thresholding, while distance estimation uses camera calibration parameters and the object's actual size to calculate depth information.

- **How it works**:
  - **Color block detection**: First, the image is segmented using a predefined color threshold (e.g., RGB range) to extract pixel regions that meet the criteria. Morphological operations (such as erosion and dilation) are then used to refine the detection results, filter noise, and connect adjacent regions. The detection result typically returns the bounding box (x, y, width, height) of the color block.
  - **Distance estimation**: Based on the PnP (Perspective-n-Point) algorithm, it uses the camera intrinsic parameter matrix (camera_matrix) and distortion coefficients (dist_coeffs) to estimate the object's position in 3D space. Combined with the known actual object dimensions (such as the actual width and height of the color block), the distance is calculated through triangulation. The formula involves projective geometry and inverse perspective transformation.
- **Effect**: This operation can detect objects of a specific color in an image in real time, draw a bounding box on the image, and display the estimated distance. It is suitable for scenarios requiring fast localization and measurement. Detection accuracy is affected by color thresholding, image quality, and camera calibration.
- **Application Scenarios**: Color block detection and distance estimation are commonly used in robotic vision (e.g., tracking colored signs), augmented reality (AR), industrial automation (e.g., detecting product defects), and drone navigation. On embedded devices, this approach combines simplicity and practicality, making it suitable for resource-constrained environments.

In practical applications, parameter adjustments (e.g., color thresholding and minimum area) can optimize detection accuracy, while camera parameters ensure the reliability of distance estimation.

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 # Custom CV extension
import ulab.numpy as np # MicroPython version of NumPy / Lightweight NumPy for
uPython
```

Setting the Image Size

```
image_shape = [480, 640] # Image 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) # Set pixel format to RGB888
```

- 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 color block detection).

Initialize the display module

```
Display.init(Display.VIRT, width=image_shape[1], height=image_shape[0],
to_ide=True, quality=50)
```

Initialize the display module, using the virtual display driver (Display.VIRT) with a resolution consistent with the image size. `to_ide=True` indicates that the image will be simultaneously transferred to the IDE for virtual display, and `quality=50` sets the image transfer quality.

Initialize the media manager and start the camera

```
MediaManager.init()
sensor.run() # Start capturing
```

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

Setting blob detection parameters

```
threshold = [123, 145, 48, 79, 44, 73] # Color thresholds [Rmin, Rmax, Gmin,
Gmax, Bmin, Bmax]
min_area = 10 # Minimum valid blob area
kernel_size = 1 # Morphological kernel size
```

- `threshold`: RGB color threshold array, used to define the color range for detection.
- `min_area`: Minimum blob area threshold, used to filter out undersized detection areas.
- `kernel_size`: Kernel size for morphological operations, used to optimize detection results.

Set the camera's intrinsic parameters and distortion coefficients

> The intrinsic parameters and distortion coefficients given here are for the GC2093 camera included with the Yabo Smart K230. Initialize the corresponding parameters at 1280*960.

```
camera_matrix = [
    1199.1495245491617, 0.0, 670.4213222250077,
    0.0, 1200.6879329729338, 475.11122810359876,
    0.0, 0.0, 1.0
]
dist_coeffs = [-0.0331624688297151, 0.04145795463721313, 0.0016964134879770974,
0.0008834973490863845, -0.05414957483933673]
dist_len = len(dist_coeffs)

# Actual ROI size (unit: cm) / Real-world size of detected ROI (cm)
roi_width_real = 3.7 # Example: Blob width 3cm / Real-world Blob width
roi_height_real = 3.7 # Example: Blob height 2.8cm / Real-world Blob height
```

- `camera_matrix`: Camera intrinsic parameter matrix, used for distance estimation.
- `dist_coeffs`: Camera distortion coefficients, used to correct for lens distortion.
- `roi_width_real` and `roi_height_real`: detect the actual physical size of the color patch, which is used to calculate the distance.

Setting the frame rate timer

```
clock = time.clock()
```

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

Image processing, color block detection, and distance estimation

```
while True:
    clock.tick()  # Start FPS timer

    img = sensor.snapshot() # Capture one frame
    img_np = img.to_numpy_ref() # Get NumPy reference to RGB data

    # Color block detection: Returns multiple color blocks [x, y, w, h, ...] /
Detect color blobs
    blobs = cv_lite.rgb888_find_blobs(image_shape, img_np, threshold, min_area,
kernel_size)

    if len(blobs) > 0:
        # Get first blob's bounding box
        roi = [blobs[0], blobs[1], blobs[2], blobs[3]]

        # Estimate distance via PnP
        distance = cv_lite.rgb888_pnp_distance(
            image_shape, img_np, roi,
            camera_matrix, dist_coeffs, dist_len,
            roi_width_real, roi_height_real
        ) // 2

        # Draw bounding box and distance text
        img.draw_rectangle(roi[0], roi[1], roi[2], roi[3], color=(255, 0, 0),
thickness=2)
        img.draw_string_advanced(roi[0], roi[1] - 20, 32, str(distance), color=
(255, 0, 0))

    # Display image to IDE
```

```
    Display.show_image(img)

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

    # Garbage collection
    gc.collect()
```

- **Image Capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Block Detection**: Call `cv_lite.rgb888_find_blobs()` to detect blobs and return a list of blobs.
- **Distance Estimation**: Calculate the distance of the detected blobs using `cv_lite.rgb888_pnp_distance()`.
- **Drawing and Display**: Draw the blob bounding box and distance text 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 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()
```

Parameter adjustment instructions

- `threshold`: RGB color threshold array [Rmin, Rmax, Gmin, Gmax, Bmin, Bmax], used to define the color range for detection. Adjust the value to match the target color (for example, adjust the threshold according to the ambient light). It is recommended to use image analysis tools to test the threshold range.
- `min_area`: Minimum color block area. Larger values (such as 10-50) filter more small noise, and smaller values make detection more sensitive. Adjust according to image resolution and target size.
- `kernel_size`: Morphological kernel size. Larger values (e.g., 1-5) enhance the morphological operation but may blur the edges of the color patches. It is recommended to start with 1 for testing.
- `roi_width_real` **and** `roi_height_real`: The actual physical dimensions of the color patches (in centimeters). Adjusted based on real-world object measurements to improve distance estimation accuracy.
- `camera_matrix` **and** `dist_coeffs`: Camera calibration parameters. Adjust only when the camera changes. Be sure to use the calibration tool to obtain accurate values to avoid distance estimation errors.