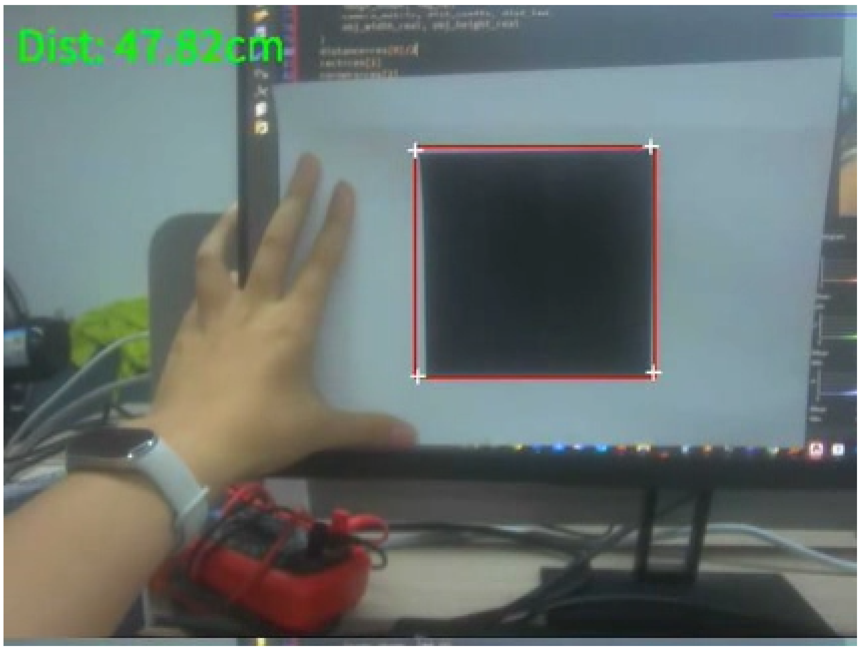
Finding Rectangles and Measuring Distances

Example Results

Run this section's example code [Source Code/06.cv_lite/27.rgb888_pnp_distance_from_corners.py]

In this section, we will use the `cv_lite` extension module to implement image contour detection on an embedded device and use the PnP algorithm to estimate the distance of the detected contours.



Principle Explanation

What are Image Contour Detection and PnP Distance Estimation?

Image contour detection is a computer vision technique used to extract the boundaries or edge contours of objects in an image. PnP distance estimation, based on the principle of perspective projection, combines camera parameters and the actual size of the object to calculate the distance of the object in three-dimensional space. This entire process is often used for real-time object localization and depth estimation.

- **Operation Principle**:
  - **Contour Detection**: First, contours are extracted from the image using edge detection (such as Canny) or threshold segmentation, followed by filtering for regular shapes (such as rectangles). In this code, contour detection may be embedded in a Perspective-n-Point (PnP) function, which processes the image to find key points (such as four corners).
  - **PnP Distance Estimation**: The Perspective-n-Point (PnP) algorithm uses known 3D points (based on the object's actual dimensions) and corresponding 2D image points (detected contour corners), combined with the camera's intrinsic parameter matrix and distortion coefficients, to calculate the object's pose and distance by minimizing the reprojection error. The formula involves solving the PnP problem, typically using the EPnP or DLT algorithms.
- **Effect**: This operation detects contours in the image and draws bounding boxes, corner points, and distance text on the image. It is suitable for scenarios requiring precise positioning. The detection results are affected by image quality, lighting conditions, and parameters, and may contain noise or errors.
- **Application Scenarios**: Contour detection and PnP distance estimation are widely used in robotic navigation (e.g., obstacle avoidance), augmented reality (AR), object tracking, and industrial inspection (e.g., measuring product dimensions). On embedded devices, this approach combines efficiency and accuracy, making it suitable for resource-constrained environments.

In practical applications, the accuracy of camera parameters and the measurement of the actual object size are critical to ensure reliable distance estimation.

Code Overview

Importing Modules

```python
import time, os, gc
from machine import Pin
from media.sensor import *
from media.display import *
from media.media import *
import _thread
import cv_lite # Requires implementation of the corresponding native C interface
import ulab.numpy as np
```

Setting the Image Size

```
image_shape = [480, 640]
```

Define the image resolution to 480x640 (height x width). The camera and display modules will be initialized based on this size later.

Initialize the camera (RGB888 format)

```
sensor = Sensor(id=2,width=1280,height=960,fps=90)
sensor.reset()
# Set the resolution of the captured image
sensor.set_framesize(w=image_shape[1], h=image_shape[0],chn=CAM_CHN_ID_0)
sensor.set_pixformat(Sensor.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 contour 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 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 intrinsic parameters and distortion coefficients

> The camera intrinsic parameters given here are obtained after calibration of the GC2093 camera on the Yabo Intelligent K230.

```
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, -0.054149574839336731]
dist_len = len(dist_coeffs)

# ------------------------------
# Actual object size (in cm)
# ------------------------------
obj_width_real = 12.8
obj_height_real = 12.8
```

- `camera_matrix`: Camera intrinsic parameter matrix, used for PnP distance estimation.
- `dist_coeffs`: Camera distortion coefficients, used to correct for lens distortion.
- `obj_width_real` and `obj_height_real`: Actual physical size of the detected object (in cm), used for distance calculation.

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, contour detection, and distance estimation

```
while True:
    clock.tick()

    img = sensor.snapshot()
    img_np = img.to_numpy_ref()

    # Distance estimation (via contour + PnP)
    res = cv_lite.rgb888_pnp_distance_from_corners(
        image_shape, img_np,
        camera_matrix, dist_coeffs, dist_len,
        obj_width_real, obj_height_real
    )
    distance=res[0]/2
    rect=res[1]
    corners=res[2]

    # If distance estimation is successful
    if distance > 0:
        img.draw_string_advanced(10, 10, 32, "Dist: %.2fcm" % distance, color=
(0, 255, 0))
        img.draw_rectangle(rect[0], rect[1], rect[2], rect[3], color=(255, 0,
0), thickness=2)
        img.draw_cross(corners[0][0],corners[0][1],color=
(255,255,255),size=5,thickness=2)
        img.draw_cross(corners[1][0],corners[1][1],color=
(255,255,255),size=5,thickness=2)
        img.draw_cross(corners[2][0],corners[2][1],color=
(255,255,255),size=5,thickness=2)
        img.draw_cross(corners[3][0],corners[3][1],color=
(255,255,255),size=5,thickness=2)
    else:
        img.draw_string_advanced(10, 10, 32, "No Rect Found", color=(255, 0, 0))

    # Display image
    Display.show_image(img)

    print("contour_pnp:", clock.fps())
    # print("Distance:", distance)
    gc.collect()
```

- **Image Capture**: Acquire a frame of image using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.

- **Contour Detection and Distance Estimation**: Call `cv_lite.rgb888_pnp_distance_from_corners()` to perform contour detection and PnP distance estimation, returning distance, rectangle, and corner information.
- **Drawing and Display**: Draw distance text, rectangles, and corner 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 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

- `camera_matrix` **and** `dist_coeffs`: Camera intrinsic parameters and distortion coefficients, only adjusted when the camera changes
- `obj_width_real` **and** `obj_height_real`: The actual physical size of the object (unit: cm), adjusted based on real object measurements; accurate values help reduce distance calculation errors, and it is recommended to recalibrate according to the target object.
- **Other parameters**: In this code, contour detection parameters (such as thresholds) may be embedded in the cv_lite function and cannot be adjusted directly. If you need to optimize the detection accuracy, it is recommended to adjust the internal settings or extended parameters of the cv_lite module through test images.