Draw key points

Example introduction

In this section, we introduce the draw_keypoints method for drawing key points

The code is located in [Source code summary / 03.Drawing / 07.keypoints_drawing.py]

What are key points?

Here, keypoints refer to feature points in an image, which are usually more prominent or unique points in the image. Specifically:

Feature point characteristics:

- These points are usually corner points, edge intersections and other locations with significant features in the image
- They are easy to identify at different viewing angles of the image
- The pixel values around these points vary greatly, making them unique

In actual applications, key points may be:

- Corner points of objects
- Feature points in texture-rich areas
- Significant intersections in patterns
- Important points on the contours of objects

Purpose:

- Target tracking
- Object recognition
- Image matching
- Motion detection

> Since finding key points consumes a lot of performance, we reduce the identified area and only identify key points in the center of the screen

API documentation

Find key points

```
image.find_keypoints([roi[, threshold=20[, normalized=False[, scale_factor=1.5[,
max_keypoints=100[, corner_detector=image.CORNER_AGAST]]]]]])
```

This function extracts ORB keypoints from the specified ROI tuple `(x, y, w, h)`. You can use the `image.match_descriptor` function to compare the two sets of keypoints to get matching regions. Returns `None` if no keypoints are found.

- `roi` is a rectangular tuple `(x, y, w, h)` of the region of interest. If not specified, the default ROI is the entire image. Only pixels within this region are operated on.
- `threshold` controls the number of keypoints to extract (in the range 0-255). For the default AGAST corner detector, this should be set to about 20; for the FAST corner detector, this should be set to about 60 to 80. The lower the threshold, the more corners are extracted.
- `normalized` is a boolean value. If `True`, turns off keypoint extraction at multiple resolutions. Set this to `True` if you don't care about handling scaling and want the algorithm to run faster.
- `scale_factor` is a floating point number greater than 1.0. Higher scale factors run faster, but the image matching is less accurate. Ideal values are between 1.35 and 1.5.
- `max_keypoints` is the maximum number of keypoints a keypoint object can hold. Lower this value if keypoint objects are too large and cause memory issues.
- `corner_detector` is the corner detector algorithm used to extract keypoints. Possible values are `image.CORNER_FAST` or `image.CORNER_AGAST`. The FAST corner detector is faster, but less accurate.

Note: This method only supports grayscale images.

Draw keypoints

```
image.draw_keypoints(keypoints[, color[, size=10[, thickness=1[, fill=False]]]])
```

Draw keypoints on the image.

- `color`: Specify the color, applicable to grayscale or RGB565 images. Default is white. For grayscale images, you can pass grayscale values (0-255); for RGB565 images, you can pass RGB565 values in reverse byte order.
- `size`: Controls the size of the keypoints.
- `thickness`: Controls the thickness of the line in pixels.
- `fill`: If True, fill the keypoints.

Returns the image object so that subsequent methods can be chained.

This method does not support compressed images and Bayer format images

Sample code

```
import sys
import uos as os
import time
from media.sensor import *
from media.display import *
from media.media import *
```

```
def init_sensor():
    """
    Initialize camera sensor with specified configuration
    使用指定配置初始化摄像头传感器
    """
    # Create sensor instance with resolution 1280x960
    # 创建分辨率为1280x960的传感器实例
    sensor = Sensor(width=1280, height=960)

    # Reset sensor to default state
    # 将传感器重置为默认状态
    sensor.reset()

    # Configure channel 1 output format to 640x480 RGB565
    # 配置通道1输出格式为640x480 RGB565
    sensor.set_framesize(width=640, height=480, chn=CAM_CHN_ID_1)
    sensor.set_pixformat(Sensor.RGB565, chn=CAM_CHN_ID_1)

    sensor.set_framesize(width=640, height=480, chn=CAM_CHN_ID_0)
    sensor.set_pixformat(Sensor.GRAYSCALE, chn=CAM_CHN_ID_0)

    return sensor

def main():
    """
    Main function to run camera preview
    运行摄像头预览的主函数
    """
    sensor = None
    roi = (220, 140, 200, 200)  # 从(110,70)开始，宽高都是100像素 Starting from
(110,70), the width and height are both 100 pixels
    try:
        # Initialize camera sensor
        # 初始化摄像头传感器
        sensor = init_sensor()

        # Initialize virtual display with 640x480 resolution
        # 初始化640x480分辨率的虚拟显示
        Display.init(Display.VIRT, width=640, height=480, to_ide=True)

        # Initialize media management
        # 初始化媒体管理
        MediaManager.init()

        # Start sensor operation
        # 启动传感器运行
        sensor.run()

        # Main loop to capture and display frames
        # 捕获和显示帧的主循环
        while True:
            # Capture frame from channel 1
            # 从通道1捕获帧
            img = sensor.snapshot(chn=CAM_CHN_ID_1)

            img_g = sensor.snapshot(chn=CAM_CHN_ID_0)
            img.draw_rectangle(roi, color=(173, 216, 230), fill=False,
thickness=3)
```

```
            keypoints = img_g.find_keypoints(
                threshold=30,
                scale_factor=1.2,
                max_keypoints=30,    # 减少特征点数量 Reduce the number of feature
points
                roi=roi              # 指定ROI区域 Specify ROI area
            )

            # 如果检测到特征点
            # If feature points are detected
            if keypoints:
                print(keypoints)
                # 在图像上绘制特征点
                # Draw feature points on the image
                img.draw_keypoints(
                    keypoints,     # 特征点列表 Feature point list
                    color=(255, 0, 0),  # 红色 red
                    size=8,             # 特征点大小 Feature point size
                    thickness=4,         # 线条粗细 Line Thickness
                    fill=True           # 填充特征点 Fill feature points
                )
            # Display captured frame
            # 显示捕获的帧
            Display.show_image(img)

    except KeyboardInterrupt:
        print("User interrupted the program")
        print("用户中断了程序")

    except Exception as e:
        print(f"An error occurred: {str(e)}")
        print(f"发生错误: {str(e)}")

    finally:
        # Cleanup section
        # 清理部分

        # Stop sensor if initialized
        # 如果传感器已初始化则停止
        if isinstance(sensor, Sensor):
            sensor.stop()

        # Deinitialize display
        # 反初始化显示
        Display.deinit()

        # Enable sleep mode
        # 启用睡眠模式
        os.exitpoint(os.EXITPOINT_ENABLE_SLEEP)
        time.sleep_ms(100)

        # Release media resources
        # 释放媒体资源
        MediaManager.deinit()

if __name__ == "__main__":
    main()
```

Example running effect

We use the small box in the middle of the screen to check the surrounding environment, and we will find that some key points in the picture are marked