Touch Display

Introduction to the results of routine experiments

> In this section, we demonstrate the use of the K230 touch screen.

The example code is located in [Source Code/02.Basic/16.touch.py]

We use CanMV IDE to open the sample code and connect K230 to the computer via USB.

Click the Run button in the lower left corner to start running the program

The initial interface is white

We move our finger on the screen and can see that black lines are drawn on the screen as the finger moves.

Routine code

Complete code

```
# Import required modules
# 导入所需的模块
import time, os, urandom, sys
•
# Import display and media related modules
# 导入显示和媒体相关模块
from media.display import *
from media.media import *
•
# Import touch sensor module
# 导入触摸传感器模块
```

```
from machine import TOUCH

# Initialize touch sensor on pin 0
# 在引脚0上初始化触摸传感器
tp = TOUCH(0)

# Define display resolution constants
# 定义显示分辨率常量
DISPLAY_WIDTH = 640
DISPLAY_HEIGHT = 480

def display_test():
    """
    Function to test display and touch functionality
    测试显示和触摸功能的函数
    """
    print("display and touch test")

    # Create main background image with white color
    # 创建白色背景的主图像
    img = image.Image(DISPLAY_WIDTH, DISPLAY_HEIGHT, image.ARGB8888)
    img.clear()
    img.draw_rectangle(0, 0, DISPLAY_WIDTH, DISPLAY_HEIGHT,color=
(255,255,255),fill=True)

    # Create secondary image for drawing
    # 创建用于绘画的次要图像
    img2 = image.Image(DISPLAY_WIDTH, DISPLAY_HEIGHT, image.ARGB8888)
    img2.clear()

    # Initialize display with ST7701 driver
    # 使用ST7701驱动初始化显示器
    Display.init(Display.ST7701, width = DISPLAY_WIDTH, height = DISPLAY_HEIGHT,
to_ide = True)
    # Initialize media manager
    # 初始化媒体管理器
    MediaManager.init()

    try:
        # Variables to store previous touch coordinates
        # 存储上一次触摸坐标的变量
        last_x = None
        last_y = None
        while True:
            os.exitpoint()
            # Read touch point data
            # 读取触摸点数据
            point = tp.read(1)

            if len(point):
                print(point)
                pt = point[0]
                # Handle touch events (down or move)
                # 处理触摸事件（按下或移动）
                if pt.event == 0 or pt.event == TOUCH.EVENT_DOWN or pt.event ==
TOUCH.EVENT_MOVE:
                    if((last_x is not None) and (last_y is not None) and
pt.event is not 2):
```

```
                        # Draw line between previous and current touch points
                        # 在上一个触摸点和当前触摸点之间画线
                        img2.draw_line(last_x,last_y,pt.x, pt.y, color=(0,0,0),
thickness = 5)
                        Display.show_image(img2, layer = Display.LAYER_OSD2,
alpha = 128)
                    last_x = pt.x
                    last_y = pt.y
            # Update display with background image
            # 更新显示背景图像
            Display.show_image(img)

            time.sleep(0.05)
    except KeyboardInterrupt as e:
        print("user stop: ", e)
    except BaseException as e:
        print(f"Exception {e}")

    # Cleanup and deinitialize display
    # 清理并反初始化显示器
    Display.deinit()
    os.exitpoint(os.EXITPOINT_ENABLE_SLEEP)
    time.sleep_ms(100)
    # Release media resources
    # 释放媒体资源
    MediaManager.deinit()

if __name__ == "__main__":
    # Enable exit points and run display test
    # 启用退出点并运行显示测试
    os.exitpoint(os.EXITPOINT_ENABLE)
    display_test()
```

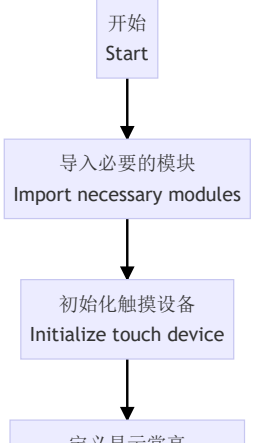
Code structure

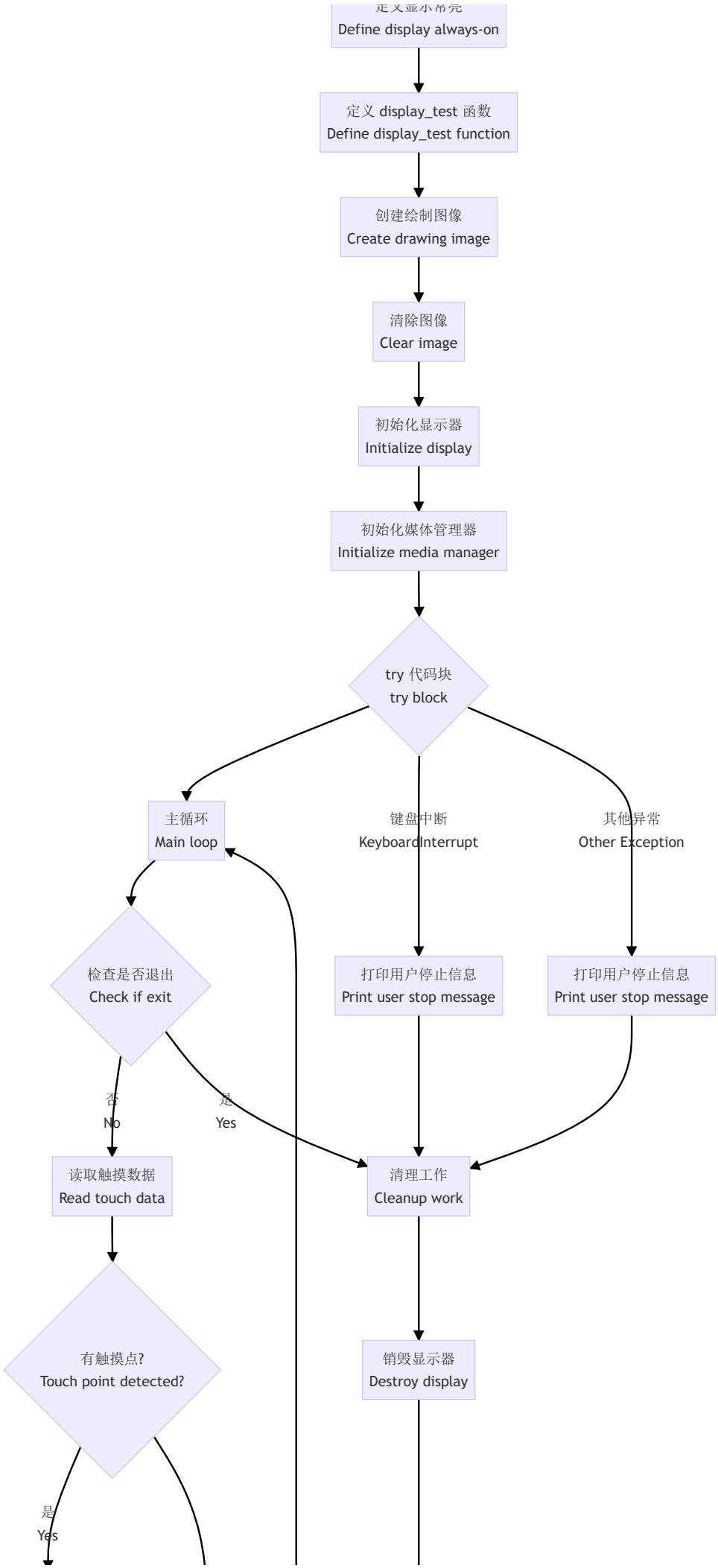
1. Basic settings:
    - Import necessary modules
    - Initialize the touch device
    - Defining display constants
    - Creating and initializing an image
    - Initialize the display and media manager
2. In the main loop:
    - Check if you need to exit
    - Reading touch data
    - If there is a touch point:
        - Checking the touch event type
        - If there is a previous coordinate point, draw a connecting line
        - Update coordinates
        - Displaying images
    - Short sleep
3. Exception handling:

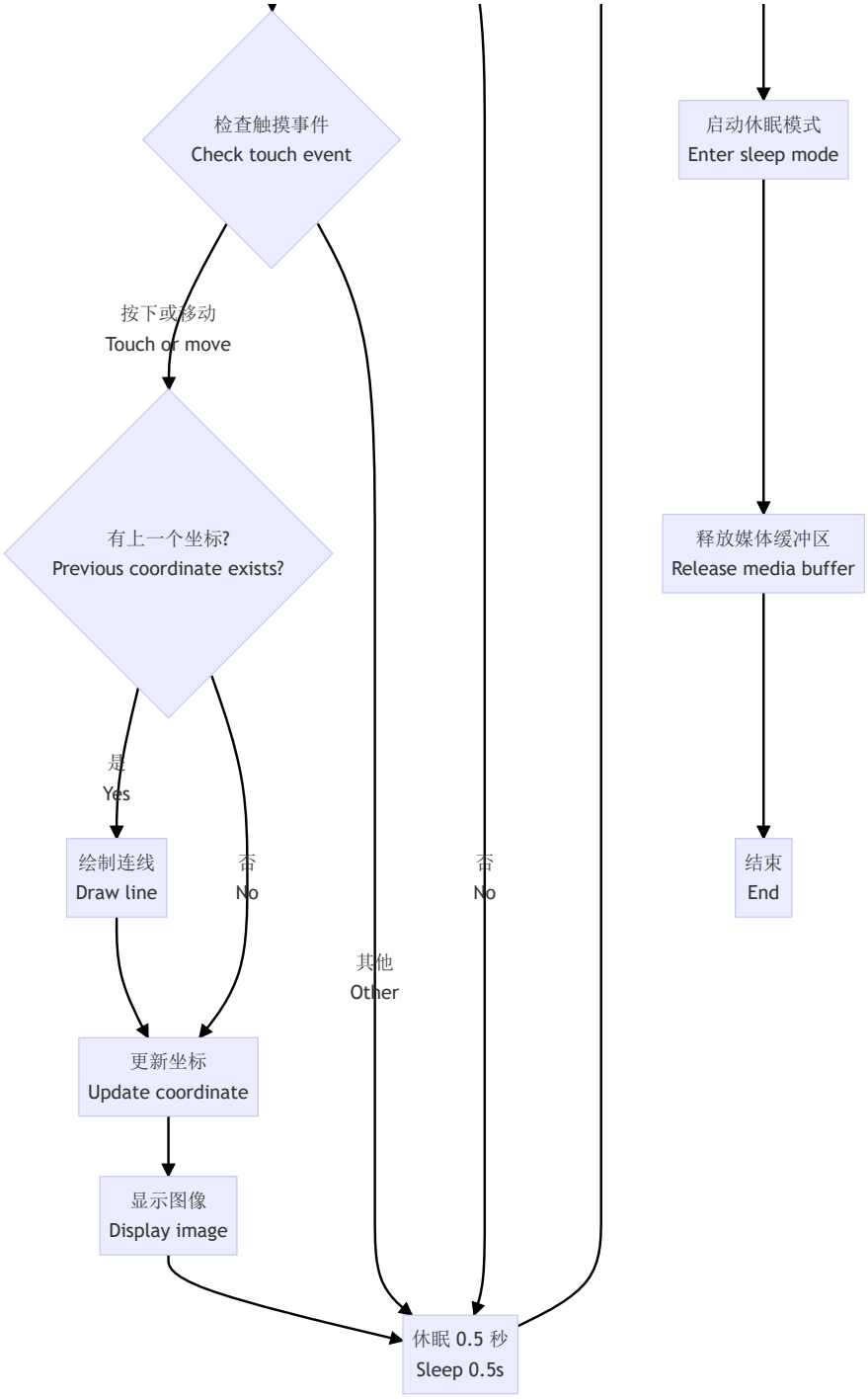
- Handling keyboard interrupts
- Handling other exceptions

4. Cleaning:
    - Destroy the display
    - Enable Hibernation Mode
    - Release the media buffer

The complete flow chart is as follows:




定义显示常亮
Define display always-on
定义 display_test 函数
Define display_test function
创建绘制图像
Create drawing image
清除图像
Clear image
初始化显示器
Initialize display
初始化媒体管理器
Initialize media manager
try 代码块
try block
主循环
Main loop
键盘中断
KeyboardInterrupt
其他异常
Other Exception
检查是否退出
Check if exit
打印用户停止信息
Print user stop message
打印用户停止信息
Print user stop message
否
No
是
Yes
读取触摸数据
Read touch data
清理工作
Cleanup work
有触摸点?
Touch point detected?
销毁显示器
Destroy display
是
Yes


检查触摸事件
Check touch event
按下或移动
Touch or move
有上一个坐标?
Previous coordinate exists?
是
Yes
否
No
绘制连线
Draw line
否
No
其他
Other
更新坐标
Update coordinate
显示图像
Display image
休眠 0.5 秒
Sleep 0.5s
启动休眠模式
Enter sleep mode
释放媒体缓冲区
Release media buffer
结束
End

TOUCH Related Documents

> The following content is selected from CanMV K230 TOUCH API
>
> 2.16 TOUCH Module API Manual — CanMV K230

1. Overview

The touch module is based on the RTT touch framework and supports single-point and multi-point capacitive touch screens and resistive touch screens.

2. API Introduction

The TOUCH class is located `machine` under the module.

Example

```
from machine import TOUCH

# Instantiate TOUCH device 0
tp = TOUCH(0)
# Get TOUCH data
p = tp.read()
print(p)
# Print touch point coordinates
# print(p[0].x)
# print(p[0].y)
# print(p[0].event)
```

Constructor

```
# when index is 0
touch = TOUCH(index, type = TOUCH.TYPE_CST328, rotate = -1)

# when index is 1
touch = TOUCH(index, type = TOUCH.TYPE_CST328, rotate = -1, range_x = -1,
range_y = -1, i2c : I2C = None, slave_addr = None, rst : Pin = None, int : Pin =
None)
```

parameter

- `index`: `TOUCH` Device number, when it is `0`, it means using the touch screen provided by the system, when `1` it is , it means using `CanMV` the proprietary touch driver
- `type`: Touch drive type, for specific definition, refer to touch type
- `rotate`: The rotation of panel output coordinates and screen coordinates, the value range is [0-3], specifically defines the reference coordinate rotation .
- `range_x`: `index=1` Valid when the maximum width of the touch output coordinates is
- `range_y`: `index=1` Valid when the maximum height of the touch output coordinates is
- `i2c`: `index=1` Valid when touching `I2C` the bus object
- `slave_addr`: `index=1` Valid when the slave address of the touch chip is selected. If not passed in, the driver default value will be used.
- `rst`: `index=1` Valid when touching the reset pin object
- `int`: `index=1` Valid when touching the interrupt pin object, currently not supported

`read` method

```
TOUCH.read([count])
```

Get touch data.

parameter

- `count`: The maximum number of touch points to read. The value range is [0:10]. The default value is 0, which means reading all touch points.

Return Value

Returns the touch point data as a tuple `([tp[, tp...]])`, where each `tp` is an `touch_info` instance of the class.

`deinit` method

```
TOUCH.deinit()
```

Release TOUCH resources.

parameter

none

Return Value

none

3. TOUCH_INFO Class

The TOUCH_INFO class is used to store information about touch points, which can be accessed by users through related read-only properties.

- `event`: Event code, refer to touch event for details .
- `track_id`: Touch ID, for multi-touch.
- `width`: Contact width.
- `x`: The x coordinate of the touch point.
- `y`: The y coordinate of the touch point.
- `timestamp`: Timestamp of the touch point.

4. Constants

4.1 Touch Events

- `EVENT_NONE`: No events.
- `EVENT_UP`: Touch and then lift.
- `EVENT_DOWN`: Touch to start.
- `EVENT_MOVE`: Touch and press to move.

4.2 Coordinate Rotation

- `ROTATE_0`: The coordinates are not rotated.
- `ROTATE_90`: Rotate the coordinates 90 degrees.
- `ROTATE_180`: Rotate the coordinates 180 degrees.
- `ROTATE_270`: Coordinates rotate 270 degrees.

4.3 Touch Type

- `TYPE_CST328`: `CanMV` Proprietary touch driver
- `TYPE_CST226SE`: `CanMV` Proprietary touch driver
- `TYPE_GT911`: `CanMV` Proprietary touch driver