4. K230 Color Line Patrol

This section will introduce in detail how to use K230 combined with PID algorithm to realize line patrol control of the car.

> Note: In this section, we only talk about the K230 part
>
> For the complete code, please refer to: [Source Code/05.Color/04.find_line.py]

1. Environment Preparation

- **Hardware**:
    - YAHBOOM K230 Vision Module
    - Motor Drive Module
    - Battery Power
- **Software**:
    - CanMV IDE Environment

2. Code Structure Overview

The code is mainly divided into the following parts:

1. **Library Import**: Import the required modules.
2. **Constant Definition**: Define display parameters, LAB color space threshold (that is, the color of the line patrol track), PID parameters.
3. **Function Function**: Including sensor initialization, PID calculation, image processing, etc.
4. **Main loop**: Get image, process color, calculate motor speed, and display information.

3. Code explanation

3.1 Import library

```
import time, os, sys
from media.sensor import *
from media.display import *
from media.media import *
from ybUtils.YbUart import YbUart

uart = YbUart(9600)
```

Here, the necessary libraries are imported, including the time module, media sensor, display, and serial communication libraries. `YbUart` is used for serial communication with the motor controller.

3.2 Define parameters

```
DISPLAY_WIDTH = 640   # LCD显示宽度 LCD display width
DISPLAY_HEIGHT = 480  # LCD显示高度 LCD display height

THRESHOLDS = [
    ((21, 33, -15, 9, -9, 6)),    # 黑线 Black line
    ((40, 86, -44, -20, -24, 25)), # 绿 Green
]

KP = 1
KI = 0.1
KD = 0.2

BASE_SPEED = 300
SCREEN_CENTER = DISPLAY_WIDTH // 2

prev_error = 0
integral = 0
```

- `DISPLAY_WIDTH` and `DISPLAY_HEIGHT` define the display size of the LCD.
- `THRESHOLDS` defines the thresholds of the LAB color space, which is used to identify objects of different colors.
- The PID control parameters `KP`, `KI`, `KD` are used to adjust the response of the controller.
- `BASE_SPEED` sets the base speed of the car.

3.3 Initialization function

```
def init_sensor():
    sensor = Sensor(width=1280, height=960)
    sensor.reset()
    sensor.set_framesize(width=DISPLAY_WIDTH, height=DISPLAY_HEIGHT)
    sensor.set_pixformat(Sensor.RGB565)
    return sensor

def init_display():
    Display.init(Display.ST7701, to_ide=True)
    MediaManager.init()
```

- `init_sensor()` is used to initialize the camera sensor, set the resolution and pixel format.
- `init_display()` initializes the display module.

3.4 PID calculation

```
def calculate_pid(target, current):
    global prev_error, integral

    error = target - current
    integral += error
    derivative = error - prev_error

    if integral > 100:
        integral = 100
    elif integral < -100:
        integral = -100

    output = KP * error + KI * integral + KD * derivative

    left_speed = BASE_SPEED - output
    right_speed = BASE_SPEED + output

    prev_error = error

    return int(left_speed), int(right_speed)
```

This function is used to calculate the PID control output:

- Calculate the current error, integral and derivative.
- Limit the maximum value of the integral to prevent integral saturation.
- Calculate the speed of the left and right motors.

3.5 Processing images

```
def process_blobs(img, blobs, color):
    if not blobs:
        uart.send("$0,0#")
        return

    largest_blob = max(blobs, key=lambda b: b[4])

    target_x = SCREEN_CENTER
    current_x = largest_blob[0] + largest_blob[2] // 2
    left_speed, right_speed = calculate_pid(target_x, current_x)

    uart.send(f"${left_speed},{right_speed}#")

    img.draw_rectangle(largest_blob[0:4], color=color, thickness=4)
    img.draw_cross(largest_blob[0] + largest_blob[2]//2, largest_blob[1] +
largest_blob[3]//2, color=color, thickness=2)
    img.draw_line(SCREEN_CENTER, 0, SCREEN_CENTER, DISPLAY_HEIGHT, color=(0,
255, 0), thickness=1)
    img.draw_line(current_x, largest_blob[1], current_x, largest_blob[1] +
largest_blob[3], color=(255, 0, 0), thickness=2)
```

- `process_blobs()` processes the detected blobs.
- Find the largest blob and calculate its position, and then calculate the PID output and send it to the motor.

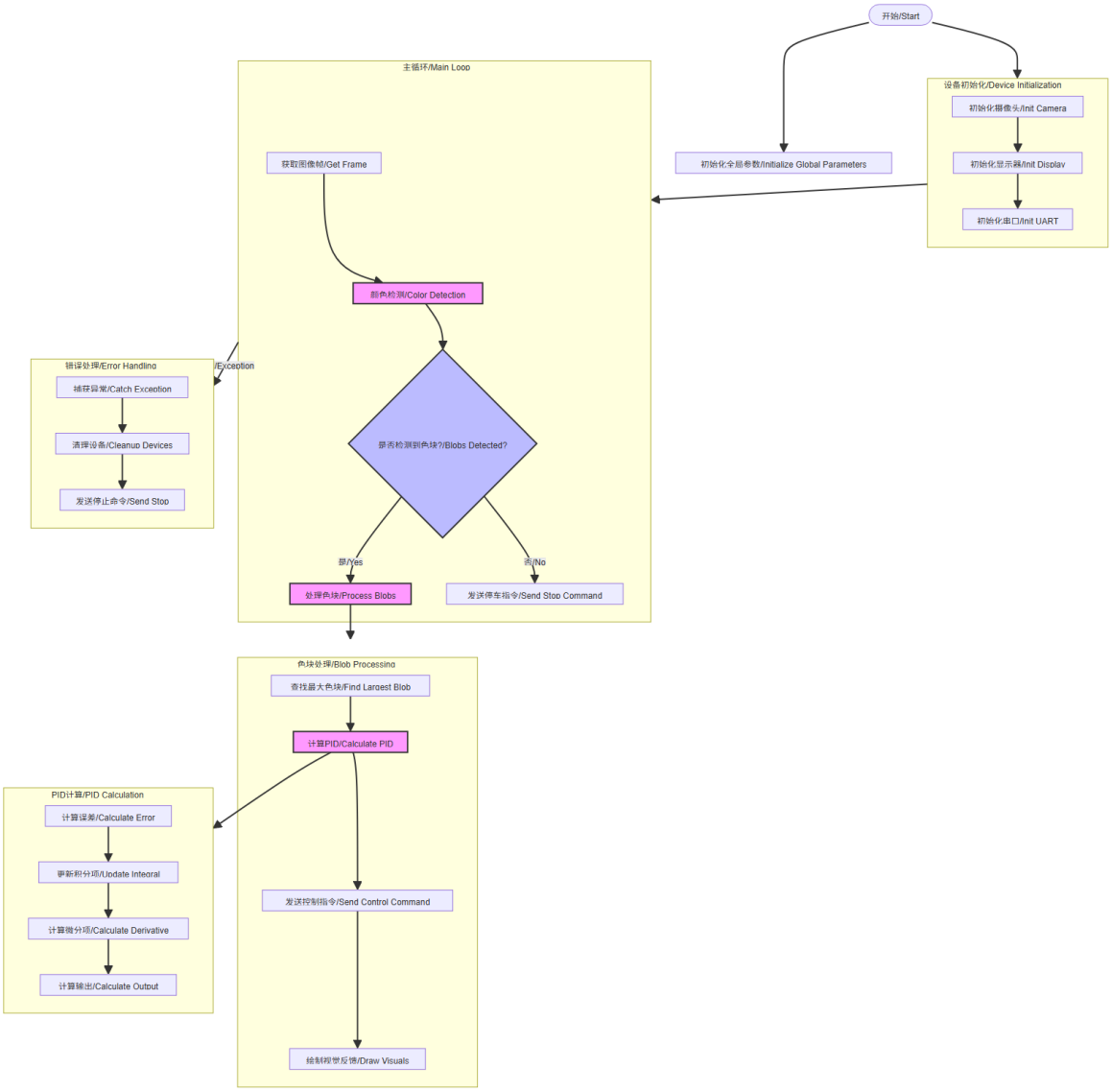
- Draw the blob and reference lines.

3.6 Main loop

```
def main():
    try:
        sensor = init_sensor()
        init_display()
        sensor.run()

        clock = time.clock()
        color_index = 1
        threshold = THRESHOLDS[color_index]
        detect_color = get_closest_rgb(threshold)

        while True:
            clock.tick()
            img = sensor.snapshot()
            blobs = img.find_blobs([threshold], roi=(0, DISPLAY_HEIGHT//2,
DISPLAY_WIDTH, DISPLAY_HEIGHT//2))
            if blobs:
                process_blobs(img, blobs, detect_color)
            else:
                uart.send("$0,0#")

            fps = clock.fps()
            draw_fps(img, fps)
            print(fps)

            Display.show_image(img)

    except KeyboardInterrupt as e:
        print("用户中断:", e)
    except Exception as e:
        print(f"发生错误: {e}")
    finally:
        if 'sensor' in locals() and isinstance(sensor, Sensor):
            sensor.stop()
        Display.deinit()
        MediaManager.deinit()

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

- In the `main()` function, initialize the sensor and display module and enter the main loop.
- In the loop, acquire the image and detect the specified color. Adjust the motor speed based on the detection result and display the image and FPS information.

3.7 Flowchart


开始/Start
初始化全局参数/Initialize Global Parameters
设备初始化/Device Initialization
初始化摄像头/Init Camera
初始化显示器/Init Display
初始化串口/Init UART
主循环/Main Loop
获取图像帧/Get Frame
颜色检测/Color Detection
是否检测到色块?/Blobs Detected?
是/Yes
否/No
处理色块/Process Blobs
发送停车指令/Send Stop Command
/Exception
错误处理/Error Handling
捕获异常/Catch Exception
清理设备/Cleanup Devices
发送停止命令/Send Stop
色块处理/Blob Processing
查找最大色块/Find Largest Blob
计算PID/Calculate PID
发送控制指令/Send Control Command
绘制视觉反馈/Draw Visuals
PID计算/PID Calculation
计算误差/Calculate Error
更新积分项/Update Integral
计算微分项/Calculate Derivative
计算输出/Calculate Output