main.py

```
# -*- coding: utf-8 -*-
from classify_garbage import classify_garbage
from control_led import control_led
from control_led import blink_led
from red_led import control_red_led
from flash_red_led import flash_red_led
from full_garbage import is_full
from start_classify import start_classify
import time

if __name__ == "__main__":
    #四个垃圾类型的垃圾桶
    sz = {
    '可回收物',
    '厨余垃圾',
    '有害垃圾',
    '其他垃圾',
    }
    #运行
    while(1):
        #检查垃圾桶是否已经满了
        for str in sz:
            # print(str)
            #相应的垃圾桶满了则亮红灯
            if is_full(str):
                print("red_led flash")
                control_red_led(str)
        #调用垃圾分类智能识别
        result = start_classify()
        print(result)
        if result:
            #垃圾分类结果类别
            print("start garbage_classify")
            garbage_type = classify_garbage()
    #如果对应的垃圾桶已经满了，则红灯闪烁通知用户
            if is_full(garbage_type):
                flash_red_led(garbage_type)
            #正常情况下相应类别的垃圾桶对应的灯光闪烁
            else:
                print("normal")
                control_led(garbage_type)
        time.sleep(0.5)
```

```
45.
46.
47. distance.py
48.
49. import RPi.GPIO as GPIO
50. import time
51.
52. def get_distance(GPIO_TRIGGER, GPIO_ECHO):
53.     """
54.     获取距离的函数
55.     :param GPIO_TRIGGER: 超声波传感器的 TRIG 引脚
56.     :param GPIO_ECHO: 超声波传感器的 ECHO 引脚
57.     :return: 距离（单位：厘米）
58.     """
59.     # 设置 GPIO 模式为 BCM
60.     GPIO.setmode(GPIO.BCM)
61.
62.     # 设置引脚模式
63.     GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
64.     GPIO.setup(GPIO_ECHO, GPIO.IN)
65.
66.     # 确保 TRIG 引脚为低电平
67.     GPIO.output(GPIO_TRIGGER, False)
68.     time.sleep(2)  # 等待传感器稳定
69.
70.     # 发送超声波信号
71.     GPIO.output(GPIO_TRIGGER, True)
72.     time.sleep(0.00001)  # 发送 10 微秒的脉冲
73.     GPIO.output(GPIO_TRIGGER, False)
74.
75.     # 记录开始时间
76.     start_time = time.time()
77.     stop_time = time.time()
78.
79.     # 等待 ECHO 引脚变为高电平
80.     while GPIO.input(GPIO_ECHO) == 0:
81.         start_time = time.time()
82.
83.     # 等待 ECHO 引脚变为低电平
84.     while GPIO.input(GPIO_ECHO) == 1:
85.         stop_time = time.time()
86.
87.     # 计算时间差
88.     time_elapsed = stop_time - start_time
```

```
    # 声速为 34300 cm/s，计算出距离
    distance = (time_elapsed * 34300) / 2

    # GPIO.cleanup()
    return distance


full_garbage.py

import RPi.GPIO as GPIO
import time
from distance import get_distance

# 设置 GPIO 模式为 BCM
GPIO.setmode(GPIO.BCM)

# 定义 GPIO 引脚
GPIO_TRIGGER_1 = 7
GPIO_ECHO_1 = 8

GPIO_TRIGGER_2 = 7
GPIO_ECHO_2 = 8

GPIO_TRIGGER_3 = 7
GPIO_ECHO_3 = 8

GPIO_TRIGGER_4 = 7
GPIO_ECHO_4 = 8

FULL_THRESHOLD = 10

def is_full(garbage_type):
    #print("enter in is_full")
    if garbage_type == "可回收物":
        distance = get_distance(GPIO_TRIGGER_1, GPIO_ECHO_1)
    elif garbage_type == "厨余垃圾":
        distance = get_distance(GPIO_TRIGGER_2, GPIO_ECHO_2)
    elif garbage_type == "有害垃圾":
        distance = get_distance(GPIO_TRIGGER_3, GPIO_ECHO_3)
    elif garbage_type == "其他垃圾":
        distance = get_distance(GPIO_TRIGGER_4, GPIO_ECHO_4)
    else:
        return False

```

```
    if distance <= FULL_THRESHOLD:
        #print("The trash can is full!")
        return True
    else:
        #print("The trash can is not full")
        return False


start_classify.py

import RPi.GPIO as GPIO
import time
from distance import get_distance

# 定义 GPIO 引脚
GPIO_TRIGGER = 7
GPIO_ECHO = 8


def start_classify():
    """
    检测距离：
    1. 如果距离超过 1m，返回 False 并重置计时器。
    2. 如果 30cm 到 1m 之间且停留超过 5 秒，返回 True。
    3. 如果停留超过 1 分钟，返回 False 并重置计时器。
    """
    distance = get_distance(GPIO_TRIGGER, GPIO_ECHO)
    print(f"distance is: {distance:.2f} cm")

    # 初始化变量
    if not hasattr(start_classify, "last_detection_time"):
        start_classify.last_detection_time = None  # 静态变量，记录最后检测时间
        start_classify.timer_reset = False  # 用于标记是否需要重置计时器
    person_detected = False  # 默认没有检测到人

    # 处理距离大于 1m 的情况
    if distance > 100:
        start_classify.last_detection_time = None  # 距离超过 1m，重置计时器
        start_classify.timer_reset = True
        print("Distance is greater than 1m, resetting timer.")
        return False  # 返回 False，重置计时器

    # 处理 20cm 到 1m 之间的情况
    if 20 <= distance <= 100:
```

```
        print(f"Person detected within {distance} cm!")
        if start_classify.last_detection_time is None:
            start_classify.last_detection_time = time.time()  # 记录检测到人的时间
            elapsed_time = 0
        else:
            elapsed_time = time.time() - start_classify.last_detection_time
            print(f"Person has been detected for {elapsed_time:.2f} seconds.")
            if elapsed_time > 60:  # 如果停留超过 1 分钟，返回 False
                print("Person has been detected for more than 1 minute!")
                start_classify.last_detection_time = None  # 重置计时器
                return False

        if elapsed_time > -1:  # 如果停留超过 5 秒
            print("Person has been detected for more than 5 seconds!")
            # classify_garbage()  # 调用垃圾分类函数
            start_classify.last_detection_time = None  # 重置计时器
            return True  # 返回 True，表示检测到人且停留超过 5 秒

    if distance < 20:
        print("distance too close!")
        return False  # 没有检测到人或时间不足，返回 False


control_led.py

import RPi.GPIO as GPIO
import time


def blink_led(led_pin, duration, interval=0.5):
    """闪烁 LED 灯"""
    end_time = time.time() + duration
    while time.time() < end_time:
        GPIO.output(led_pin, True)
        time.sleep(interval)
        GPIO.output(led_pin, False)
        time.sleep(interval)


def control_led(garbage_type):
    print(garbage_type)
    # 设置 GPIO 的编号模式
    GPIO.setmode(GPIO.BCM)
    # GPIO.setwarnings(False)
```

```
221.
222.    # 定义 LED 对应的 GPIO 引脚
223.    led_pins = {
224.        '可回收物': 17,
225.        '厨余垃圾': 27,
226.        '有害垃圾': 22,
227.        '其他垃圾': 9,
228.    } # 请根据实际连接的引脚修改
229.
230.    # 设置 GPIO 引脚为输出模式，并初始化为关闭状态
231.
232.    """控制 LED 灯闪烁 3 秒后常亮 10 秒"""
233.    if garbage_type in led_pins:
234.        led_pin = led_pins[garbage_type]
235.        GPIO.setup(led_pin, GPIO.OUT)
236.        GPIO.output(led_pin, False)  # 初始状态为关闭
237.        blink_led(led_pin, 3)  # 闪烁 3 秒
238.        GPIO.output(led_pin, True)  # 常亮
239.        time.sleep(5)  # 保持 5 秒
240.        GPIO.output(led_pin, False)  # 关闭 LED
241.        GPIO.cleanup()
242.        return
243.    else:
244.        print("未知垃圾类型")
245.
246.
247. red_led.py
248.
249. import RPi.GPIO as GPIO
250. import time
251.
252. # 定义白灯和红灯对应的 GPIO 引脚
253. white_led_pins = {
254.    '可回收物': 17,
255.    '厨余垃圾': 27,
256.    '有害垃圾': 22,
257.    '其他垃圾': 9,
258. }
259.
260. red_led_pins = {
261.    '可回收物': 10,
262.    '厨余垃圾': 13,
263.    '有害垃圾': 19,
264.    '其他垃圾': 26,
```

```
}


def control_red_led(garbage_type):
    """当垃圾桶满时调用此函数，让此垃圾桶对应的红灯常亮"""
    # 设置 GPIO 的编号模式
    # print(garbage_type)
    GPIO.setmode(GPIO.BCM)
    red_pin = red_led_pins[garbage_type]
    # print(red_pin)
    GPIO.setup(red_pin, GPIO.OUT)
    GPIO.output(red_pin, GPIO.HIGH)  # 点亮红灯
    # white_pin = white_led_pins[garbage_type]
    # GPIO.setup(white_pin,GPIO.OUT)
    # GPIO.output(white_pin, False)  # 确保对应的白灯关闭
    # GPIO.cleanup()


flash_red_led.py

import RPi.GPIO as GPIO
import time
from control_led import blink_led


# 定义红灯对应的 GPIO 引脚
red_led_pins = {
    '可回收物': 10,
    '厨余垃圾': 13,
    '有害垃圾': 19,
    '其他垃圾': 26
}

def flash_red_led(garbage_type):
    """当垃圾桶满时，红灯闪烁 5 秒后继续常亮"""
    # 设置 GPIO 的编号模式
    GPIO.setmode(GPIO.BCM)
    red_pin = red_led_pins[garbage_type]
    GPIO.setup(red_pin,GPIO.OUT)
    blink_led(red_pin, 5)  # 闪烁 5 秒
    GPIO.output(red_pin, True)  # 继续常亮
    # GPIO.cleanup()
```

```
309.
310.
311.
312. classify_garbage.py
313.
314. # -*- coding: utf-8 -*-
315. import cv2
316. import onnxruntime
317. import numpy as np
318. import time
319. from collections import Counter
320.
321. def blur_background(frame, blur_radius=21):
322.     """
323.     对图像进行模糊化处理，以消除背景中的人物影响。
324.     :param frame: 输入的图像帧。
325.     :param blur_radius: 模糊处理的半径，值越大，背景越模糊。
326.     :return: 模糊背景后的图像。
327.     """
328.     # 创建模糊化背景的副本
329.     background = cv2.GaussianBlur(frame, (blur_radius, blur_radius), 0)
330.
331.     # 将背景和前景进行融合，保持垃圾区域清晰
332.     foreground_mask = np.ones_like(frame) * 255  # 默认前景区域是白色
333.     frame_with_blurred_bg = cv2.bitwise_and(frame, foreground_mask)  # 仅保留前景区域
334.     blurred_frame = cv2.addWeighted(frame_with_blurred_bg, 0.7, background, 0.3, 0)  # 融合背景和前
    景
335.
336.     return blurred_frame
337.
338. def classify_garbage(video_index=0, model_path='/home/heng/Desktop/number.onnx',
339.                     categories_path='/home/heng/Desktop/categories.txt',
340.                      input_size=(320, 320), num_detections=5, initial_delay=3, interval_between_detections=1)
    :
341.     """
342.     Classify garbage into major categories using an ONNX model. Performs multiple detections and
343.     returns the most frequent result. Adds a delay before starting and between detections.
344.
345.     :param video_index: Index of the video stream (default is 0 for the primary camera).
346.     :param model_path: Path to the ONNX model file.
347.     :param categories_path: Path to the categories file containing small category names.
348.     :param input_size: The size (height, width) to resize frames for the model input.
349.     :param num_detections: Number of times to perform detection to avoid misclassification.
350.     :param initial_delay: Time to wait (in seconds) before starting the first detection.
```

```
    :param interval_between_detections: Time interval (in seconds) between each detection.
    :return: A string representing the predicted major category (e.g., "可回收物", "厨余垃圾", "有害垃圾", "其他垃圾").
    """
    # Load the ONNX model
    session = onnxruntime.InferenceSession(model_path)

    # Get the model input name
    input_name = session.get_inputs()[0].name

    # Load class names from the file
    with open(categories_path, 'r', encoding='utf-8') as f:
        class_names = [line.strip() for line in f.readlines()]

    # Map small categories to major categories
    major_categories = {
        "可回收物": [],
        "厨余垃圾": [],
        "有害垃圾": [],
        "其他垃圾": []
    }

    # Assign small categories to major categories based on prefix matching
    for name in class_names:
        for major_category in major_categories.keys():
            if name.startswith(major_category):
                major_categories[major_category].append(name)
                break

    # Open the video stream
    cap = cv2.VideoCapture(video_index)

    # Initial delay before starting detection
    print(f"Waiting for {initial_delay} seconds before starting detection...")
    time.sleep(initial_delay)

    detection_results = []

    while True:
        # Read a frame from the video stream
        ret, frame = cap.read()
        if not ret:
            print("Failed to grab frame")
            cap.release()
```

```
            return "Failed to grab frame"


    # Step 1: 对背景进行模糊化处理
        frame_with_blurred_bg = blur_background(frame)

        # Preprocess the frame
        resized_frame = cv2.resize(frame, input_size)
        input_frame = cv2.cvtColor(resized_frame, cv2.COLOR_BGR2RGB)
        input_frame = input_frame.astype(np.float32) / 255.0
        input_frame = np.expand_dims(input_frame, axis=0)
        input_frame = np.transpose(input_frame, (0, 3, 1, 2))

        # Run inference and store results
        result = session.run(None, {input_name: input_frame})
        output = result[0]

        if output is not None and output.size > 0:
            predicted_class = np.argmax(output)
            confidence = output[0][predicted_class]
            predicted_label = class_names[predicted_class]

            # Determine the major category
            major_category_result = "未知类别"
            for major_category, small_categories in major_categories.items():
                if predicted_label in small_categories:
                    major_category_result = major_category
                    break

            detection_results.append(major_category_result)
             print(f"Predicted: {predicted_label}, Major Category: {major_category_result}, Confidence: {co
    nfidence:.4f}")

        # 执行 5 次检测，达到次数就停止
        if len(detection_results) >= num_detections:
            break

        # Wait for the specified interval between detections
        #print(f"Waiting for {interval_between_detections} seconds before next detection...")
        print(f"Waiting for 1 seconds before next detection...")
        # time.sleep(interval_between_detections)
        time.sleep(1)

    # Find the most frequent result (mode) from the detections
```

```
437.    most_common_category = Counter(detection_results).most_common(1)[0][0]
438.
439.    # Release resources and return the most common result
440.    cap.release()
441.    return most_common_category
```