轮 趣 科 技

OpenCV 入门及其

在 ROS2 环境下的应用

推荐关注我们的公众号获取更新资料

版本说明：

版本	日期	内容说明
V1.0	2025/3/13	第一次发布

网址：www.wheeltec.net

序言

轮趣科技（东莞）有限公司出品，该手册的应用是适配于我们 ROS2 小车产品，例如麦克纳姆轮 ROS2 小车、阿克曼 ROS2 小车等。

本手册提供的内容有 OpenCV 基本操作、OpenCV 图像处理基本概念、OpenCV 应用例程以及在 OpenCV 在 ROS2 下应用。

目录

1. OpenCV 的安装

我们提供的系统镜像都是已经安装好 ROS2 和 OpenCV 等需要的一切依赖的，用户使用我们的软硬件产品即可上手运行本教程的例程代码。

用户如果想自行安装环境的话我们这边提供两个安装命令，如下第一条命令是安装 python 环境的 opencv，第二条命令是安装 ROS2 环境下的 opencv。

安装 Python 的 OpenCV 库

pip install opencv-python

pip install opencv-contrib-python

安装 ROS2 的 cv_bridge 和 vision_opencv

sudo apt-get install ROS2-<ROS2-distro>-cv-bridge

sudo apt-get install ROS2-<ROS2-distro>-vision-opencv

将 <ros2-distro> 替换为你的 ROS 2 发行版名称（如 humble、foxy 等）。

注意：本节除 ROS2 相关的程序外，均可在 windows 环境下运行。

注意：本节所有例程 Python 程序在 linux 系统下执行如果失败，请运行以下命令后在执行程序：

sudo chmod 777 程序名.py

2. OpenCV 基本操作

本章与第 3 章【OpenCV 图像处理基本概念】为同步章节，本章主要讲 OpenCV 操作，第 3 章主要讲图像处理概念，可相互参考。

2.1 读取图片、处理图片与保存图片

① 概述

本节将会读取一张图片，并对图片进行仿射变换和在图片上画圆等操作，并一一用窗口显示出来，最后把图片保存。关于仿射变换需要一点线性代数的基础。

程序文件为【1.ImageProcess.py】运行效果如下图 2-1-1 所示。

图 2-1-1 【1.ImageProcess.py】运行效果

② 程序与解析

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为 utf-8 编码

import cv2 #引用 OpenCV 功能包
import numpy as np #引用数组功能包

#读取图片
```

```
img_origin = cv2.imread('1.ImageProcess.jpg')
img_height, img_width, img_channels = img_origin.shape #获取图片尺寸高度、宽度、通道数
cv2.imshow("windows_origin", img_origin) #窗口显示图片

#缩小图片尺寸，再次获取图片尺寸
img = cv2.resize(img_origin, (int(img_width/2),int(img_height/2)), interpolation=cv2.INTER_AREA)
img_height, img_width, img_channels = img.shape
cv2.imshow("windows", img) #显示图片缩小后的图片

Quit=0 #是否继续运行标志位
#提示停止方法
print ('Press key "Q" to stop.')

while Quit==0:
    keycode=cv2.waitKey(3) #每 3ms 刷新一次图片，同时读取 3ms 内键盘的输入
    if(keycode==ord('Q')): #如果按下“Q”键，停止运行标志位置 1，调出 while 循环，程序停止运行
        Quit=1

    ###仿射变换相关处理###

    #创建数组，用于仿射变换
    Mat1 = np.array([
        [1.6, 0, -150], #x 轴放大 1.6 倍，并平移-150
        [0, 1.6, -120]  #y 轴放大 1.6 倍，并平移-120
        ], dtype=np.float32)
    img1 = cv2.warpAffine(img, Mat1, (img_width, img_height)) #仿射变换
    cv2.imshow("windows1", img1) #窗口显示图片
    cv2.imwrite('1.ImageProcess1.jpg', img1) #保存图片

    theta = 15 * np.pi / 180 #15 度的弧度值
    #创建数组，用于仿射变换
    Mat2 = np.array([
        [1, np.tan(theta), 0], #图片绕上边框逆时针旋转 15 度
        [0, 1, 0]
        ], dtype=np.float32)
    img2 = cv2.warpAffine(img, Mat2, (img_width, img_height)) #仿射变换
    cv2.imshow("windows2", img2) #窗口显示图片
    cv2.imwrite('1.ImageProcess2.jpg', img2) #保存图片

    #创建数组，用于仿射变换
    Mat3 = np.array([
```

```
        [np.cos(theta), -np.sin(theta), 0], #绕左上角顺时针旋转 15 度
        [np.sin(theta), np.cos(theta), 0]
         ], dtype=np.float32)
    img3 = cv2.warpAffine(img, Mat3, (img_width, img_height)) #仿射变换
    cv2.imshow("windows3", img3) #窗口显示图片
    cv2.imwrite('1.ImageProcess3.jpg', img3) #保存图片

    ###仿射变换相关处理###


    ###在图片上画画###

    #画直线 cv2.line(图片, 起点坐标, 终点坐标, BGR, 线条粗细)
    cv2.line(img, (300, 150), (500,150), (255, 255, 0), 2)

    #画圆 cv2.circle(图片, 原点坐标, 半径, BGR, 线条粗细)
    cv2.circle(img, (400, 100), 75, (0, 0, 255), 5)

    #画矩形 cv2.rectangle(图片, 矩形左上角坐标, 矩形右下角, BGR, 线条粗细)
    cv2.rectangle(img, (620, 100), (700, 220), (255, 0, 0), 3)

    #创建多边形顶点坐标数组(可以有多个多边形)
    triangles = np.array([
    [(200, 100), (145, 203), (255, 203)],
    [(60, 140), (20, 197), (100, 197)]
     ])
    #画填充多边形 cv2.fillPoly(图片, 多边形顶点坐标数组, BGR)
    cv2.fillPoly(img, triangles, (0, 255, 0))

    #显示文字 cv2.putText(图片,文字,文字左下角坐标,字体显示格式,字体大小,BGR,
线条粗细)
    cv2.putText(img,'WHEELTEC',(1,30),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),
2)

    cv2.imshow("windows_draw", img) #显示画画后的图片

    ###在图片上画画###


    #while 循环里循环读取图片
    img_origin = cv2.imread('1.ImageProcess.jpg')
    img_height, img_width, img_channels = img_origin.shape #获取图片尺寸高度、宽
度、通道数
```

```
    #缩小图片尺寸，再次获取图片尺寸
    img = cv2.resize(img_origin, (int(img_width/2),int(img_height/2)),
interpolation=cv2.INTER_AREA)
    img_height, img_width, img_channels = img.shape

print ('Quitted!') #提示程序已停止
cv2.destroyAllWindows() #程序停止前关闭所有窗口
```

2.2 创建图片并对图片进行像素级操作

① 概述

本节例程将利用数组创建一个 3 通道图像，并对其像素点逐个进行赋值，还进行了截取指定区域图像的操作和 HSV、LAB 色域的转换。

程序文件为【2.PixleProcess.py】运行效果如下图 2-2-1 所示。

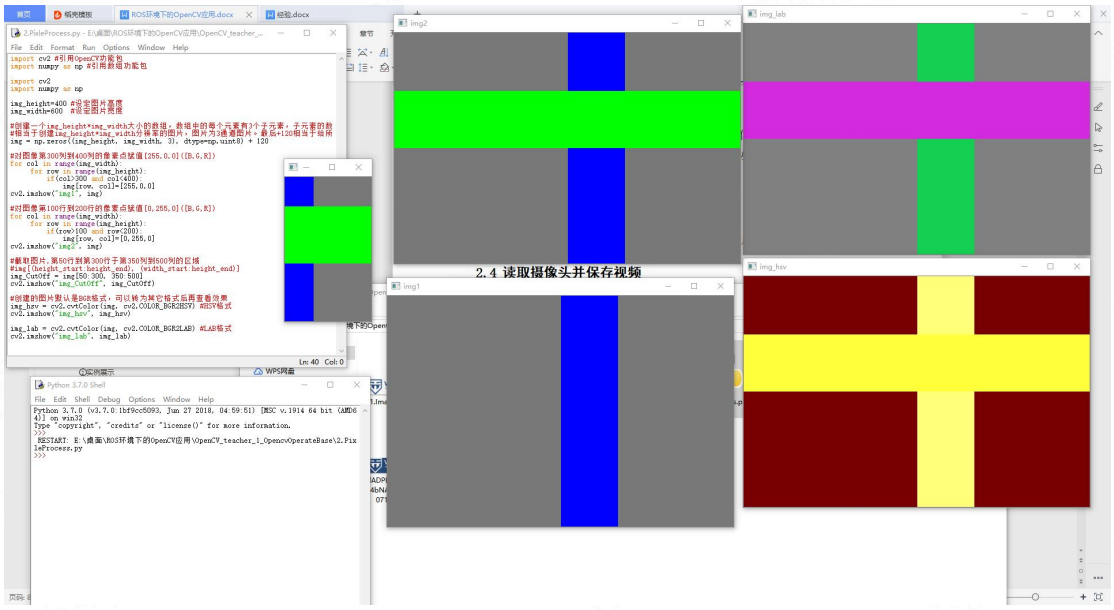

图 2-2-1【2.PixleProcess.py】运行效果

② 程序与解析

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变
量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为
utf-8 编码

import cv2 #引用 OpenCV 功能包
import numpy as np #引用数组功能包
```

```
img_height=400 #设定图片高度
img_width=600  #设定图片宽度

Quit=0 #是否继续运行标志位
#提示停止方法
print ('Press key "Q" to stop.')

while Quit==0:
    keycode=cv2.waitKey(3) #每 3ms 刷新一次图片，同时读取 3ms 内键盘的输入
    if(keycode==ord('Q')):#如果按下“Q”键，停止运行标志位置 1，调出 while 循环，
程序停止运行
        Quit=1

    #创建一个 img_height*img_width 大小的数组，数组中的每个元素有 3 个子元素，子
元素的数据类型为 uint8
    #相当于创建 img_height*img_width 分辨率的图片，图片为 3 通道图片。最后+120 相
当于给所有原始赋值 120
    img = np.zeROS2((img_height, img_width, 3), dtype=np.uint8) + 120

    #对图像第 300 列到 400 列的像素点赋值[255,0,0]([B,G,R])
    for col in range(img_width):
        for row in range(img_height):
            if(col>300 and col<400):
                img[row, col]=[255,0,0]
    cv2.imshow("img1", img)

    #对图像第 100 行到 200 行的像素点赋值[0,255,0]([B,G,R])
    for col in range(img_width):
        for row in range(img_height):
            if(row>100 and row<200):
                img[row, col]=[0,255,0]
    cv2.imshow("img2", img)

    #截取图片,第 50 行到第 300 行于第 350 列到 500 列的区域
    #img[(height_start:height_end), (width_start:height_end)]
    img_CutOff = img[50:300, 350:500]
    cv2.imshow("img_CutOff", img_CutOff)

    #创建的图片默认是 BGR 格式，可以转为其它格式后再查看效果
    img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) #HSV 格式
    cv2.imshow("img_hsv", img_hsv)

    img_lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB) #LAB 格式
    cv2.imshow("img_lab", img_lab)
```

```
print ('Quitted!') #提示程序已停止
cv2.destroyAllWindows() #程序停止前关闭所有窗口
```

2.3 读取视频流与保存视频

① 概述

本节例程将读取一个视频，并个视频加上流动水印后保存为一个新视频。

程序文件为【3.VideoRead.py】，下图 2-3-1 为保存的新视频的截图。

图 2-3-1 新视频截图

② 程序与解析

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为 utf-8 编码

import cv2 #引用 OpenCV 功能包
import numpy as np #引用数组功能包

#读取视频并获取视频帧率、分辨率、总帧数
videoCapture = cv2.VideoCapture("VideoExample.mp4")
fps=videoCapture.get(cv2.CAP_PROP_FPS)
size = (int(videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)))
```

```
totalFrames = int(videoCapture.get(7))

#创建新视频
videoWriter = cv2.VideoWriter(
    "VideoWriterExample.avi", cv2.VideoWriter_fourcc('I','4','2','0'),
    fps, size)

x=10 #水印坐标
y=10 #水印坐标
i=1
step_x=5
step_y=5
success, frame = videoCapture.read() #读取视频第一帧
print("第"+str(i)+"帧, 共"+str(totalFrames)+"帧")
while success:
    #给图片添加水印
    cv2.putText(frame, 'WHEELTEC', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
0), 2)
    cv2.imshow("frame",frame)
    videoWriter.write(frame) #给新视频添加新帧

    #水印坐标变化
    if(x>size[0]):step_x=-5
    if(x<0): step_x=5
    if(y>size[1]):step_y=-5
    if(y<0): step_y=5
    x=x+step_x
    y=y+step_y
    success, frame = videoCapture.read() #逐帧读取视频
    i=i+1
    print("第"+str(i)+"帧, 共"+str(totalFrames)+"帧")

print ('Quitted!') #提示程序已停止
cv2.destroyAllWindows() #程序停止前关闭所有窗口
```

2.4 读取摄像头并保存视频

① 概述

本节例程将读取并显示摄像头图像，在图像窗口按下按键“S”将开始录制摄像头视频，按下按键“X”将停止录制摄像头视频，按下按键“Q”将停止程序。

程序文件为【4.CameraRead.py】，下图 2-4-1 为程序运行效果图。

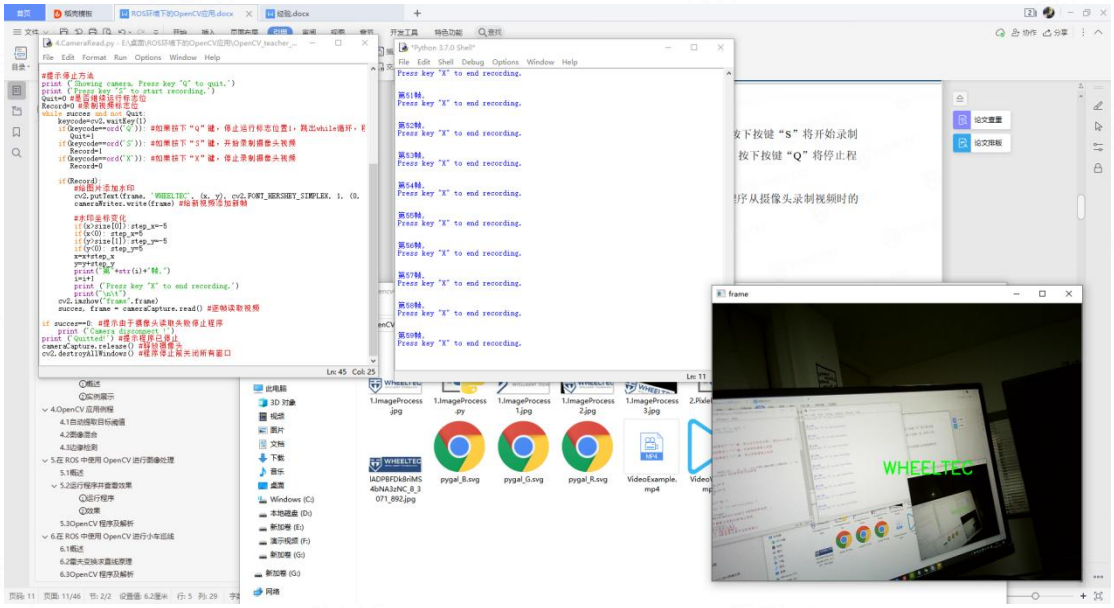

图 2-4-1 【4.CameraRead.py】运行效果

② 程序与解析

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为 utf-8 编码

import cv2 #引用 OpenCV 功能包
import numpy as np #引用数组功能包

#读取视频并获取视频帧率、分辨率
cameraCapture = cv2.VideoCapture(0)
fps=int(cameraCapture.get(cv2.CAP_PROP_FPS))
size = (int(cameraCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(cameraCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)),)

#创建新视频
cameraWriter = cv2.VideoWriter(
    "CameraWriterExample.avi", cv2.VideoWriter_fourcc('I','4','2','0'),
    fps, size)

x=10 #水印坐标
y=10 #水印坐标
i=1
```

```
step_x=5
step_y=5
succes, frame = cameraCapture.read() #读取视频第一帧

#提示停止方法
print ('Showing camera. Press key "Q" to quit.')
print ('Press key "S" to start recording.')
Quit=0 #是否继续运行标志位
Record=0 #录制视频标志位
while succes and not Quit:
    keycode=cv2.waitKey(1)
    if(keycode==ord('Q')): #如果按下“Q”键，停止运行标志位置 1，跳出 while 循环，
程序停止运行
       Quit=1
    if(keycode==ord('S')): #如果按下“S”键，开始录制摄像头视频
       Record=1
    if(keycode==ord('X')): #如果按下“X”键，停止录制摄像头视频
       Record=0

    if(Record):
        #给图片添加水印
        cv2.putText(frame, 'WHEELTEC', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 0), 2)
        cameraWriter.write(frame) #给新视频添加新帧

        #水印坐标变化
        if(x>size[0]):step_x=-5
        if(x<0): step_x=5
        if(y>size[1]):step_y=-5
        if(y<0): step_y=5
        x=x+step_x
        y=y+step_y
        print("第"+str(i)+"帧,")
        i=i+1
        print ('Press key "X" to end recording.')
        print("\n\t")
    cv2.imshow("frame",frame)
    succes, frame = cameraCapture.read() #逐帧读取视频

if succes==0: #提示由于摄像头读取失败停止程序
    print ('Camera disconnect !')
print ('Quitted!') #提示程序已停止
cameraCapture.release() #释放摄像头
cv2.destroyAllWindows() #程序停止前关闭所有窗口
```

2.5 提取并显示图像彩色直方图

① 概述

本节将读取一张彩色图像，并使用 OpenCV 的 cv2.calcHist()函数获取该图像的直方图信息，然后分别利用 Pygal、plt(matplotlib.pyplot)工具将直方图信息表现出来。

程序文件为【5.ImageHistogram.py】，下图 2-4-1 为程序运行效果图。

需要安装 pyagl 功能包：pip install pygal

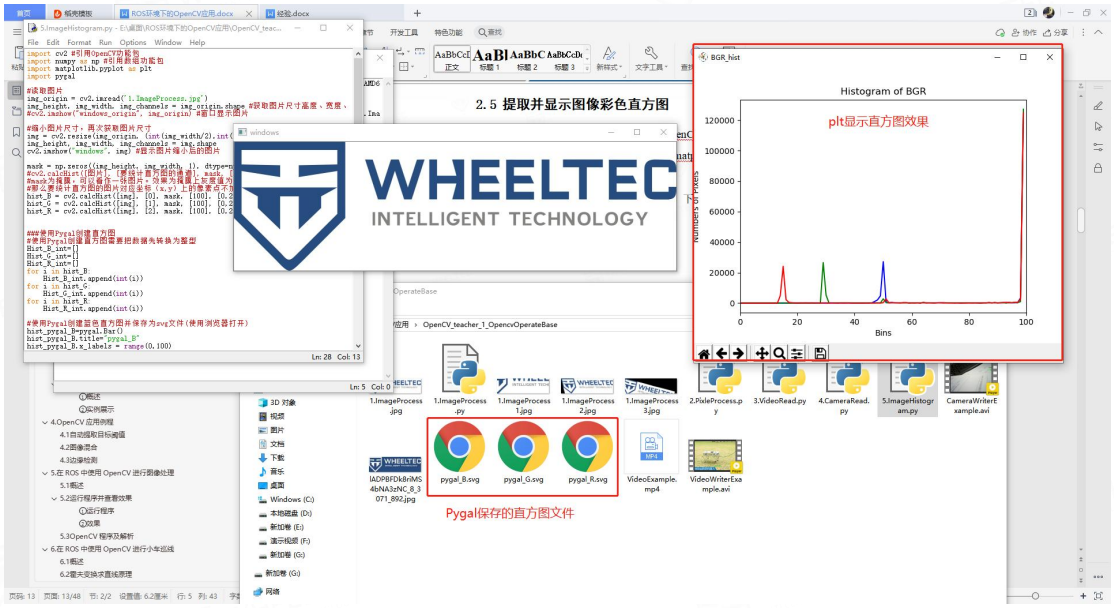

图 2-5-1 【5.ImageHistogram.py】运行效果

② 程序与解析

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为 utf-8 编码

import cv2 #引用 OpenCV 功能包
import numpy as np #引用数组功能包
import matplotlib.pyplot as plt
import pygal

#读取图片
img_origin = cv2.imread('1.ImageProcess.jpg')
```

```
img_height, img_width, img_channels = img_origin.shape #获取图片尺寸高度、宽度、通道数
#cv2.imshow("windows_origin", img_origin) #窗口显示图片

#缩小图片尺寸，再次获取图片尺寸
img = cv2.resize(img_origin, (int(img_width/2),int(img_height/2)),
interpolation=cv2.INTER_AREA)
img_height, img_width, img_channels = img.shape
cv2.imshow("windows", img) #显示图片缩小后的图片

mask = np.zeROS2((img_height, img_width, 1), dtype=np.uint8) + 1 #创建一张灰度图像
#cv2.calcHist([图片], [要统计直方图的通道], mask, [直方图横轴的坐标范围], [要统计的通道的取值范围])
#mask 为掩膜，可以看作一张图片。效果为掩膜上灰度值为 0 的像素点的坐标为（x,y），
#那么要统计直方图的图片对应坐标（x,y）上的像素点不加入直方图统计。None 则统计图片全部像素点。
hist_B = cv2.calcHist([img], [0], mask, [100], [0,256])
hist_G = cv2.calcHist([img], [1], mask, [100], [0,256])
hist_R = cv2.calcHist([img], [2], mask, [100], [0,256])


###使用 Pygal 创建直方图
#使用 Pygal 创建直方图需要把数据先转换为整型
Hist_B_int=[]
Hist_G_int=[]
Hist_R_int=[]
for i in hist_B:
    Hist_B_int.append(int(i))
for i in hist_G:
    Hist_G_int.append(int(i))
for i in hist_R:
    Hist_R_int.append(int(i))

#使用 Pygal 创建蓝色直方图并保存为 svg 文件(使用浏览器打开)
hist_pygal_B=pygal.Bar()
hist_pygal_B.title="pygal_B"
hist_pygal_B.x_labels = range(0,100)
hist_pygal_B.add("Hist_B", Hist_B_int)
hist_pygal_B.render_to_file('pygal_B.svg')

#使用 Pygal 创建绿色直方图并保存为 svg 文件(使用浏览器打开)
hist_pygal_G=pygal.Bar()
hist_pygal_G.title="pygal_G"
```

```
hist_pygal_G.x_labels = range(0,100)
hist_pygal_G.add("Hist_G", Hist_G_int)
hist_pygal_G.render_to_file('pygal_G.svg')

#使用 Pygal 创建红色直方图并保存为 svg 文件(使用浏览器打开)
hist_pygal_R=pygal.Bar()
hist_pygal_R.title="pygal_R"
hist_pygal_R.x_labels = range(0,100)
hist_pygal_R.add("Hist_R", Hist_R_int)
hist_pygal_R.render_to_file('pygal_R.svg')
###使用 Pygal 创建直方图


###使用 Plt 创建直方图
#使用 plt 创建 BGR 彩色直方图
plt.figure("BGR_hist")
plt.title("Histogram of BGR")
plt.xlabel("Bins")
plt.ylabel("Numbers of Pixels")
plt.plot(hist_B,c='blue')
plt.plot(hist_G,c='green')
plt.plot(hist_R,c='red')
plt.xlim([0, 100])
plt.show()
###使用 Plt 创建直方图
```

3. OpenCV 图像处理基本概念

3.1 图像格式

① 概述

在图像处理中图像大致可分为黑白图像、灰度图像、彩色图像三种。

黑白图像中每个像素的值只有 0(黑色)和 1(白色)，如下左图 1-1-1 所示。

灰度图像中每个像素的值则可取[0, 255]，其中 0 代表纯黑色，255 代表纯白色，如下右图 1-1-2 所示。

下图的黑白图像其实就是灰度图像经过二值化(即阈值分割)处理得来的。

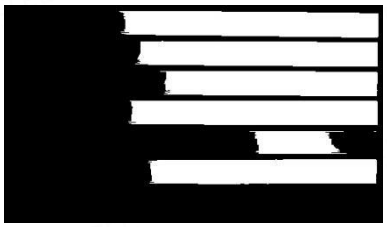

图 3-1-1 黑白图像

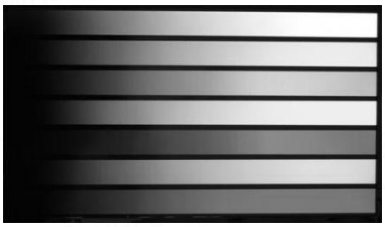

图 3-1-2 灰度图像

彩色图像一般有三个通道，按照颜色描述空间不同，彩色图像也有不同格式。

RGB 颜色空间，即红绿蓝三原色，RGB 三个通道的取值范围都为[0, 255]，彩色图像也可以看作是 3 个灰度图像的合并。

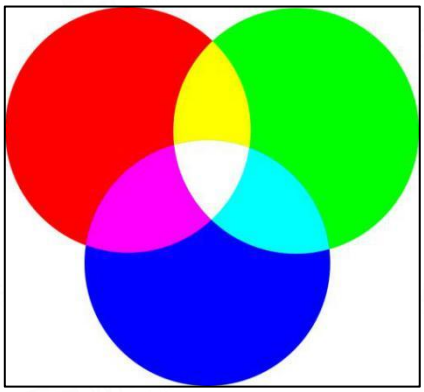

图 3-1-3 RGB 彩色图像

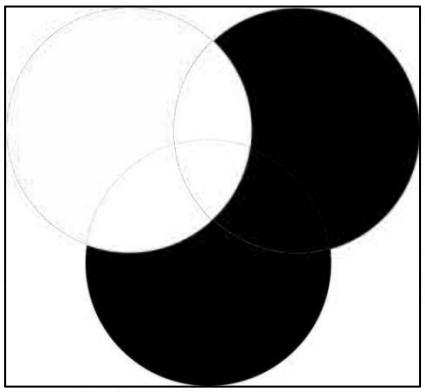

图 3-1-4 RGB 彩色图像的 R 通道

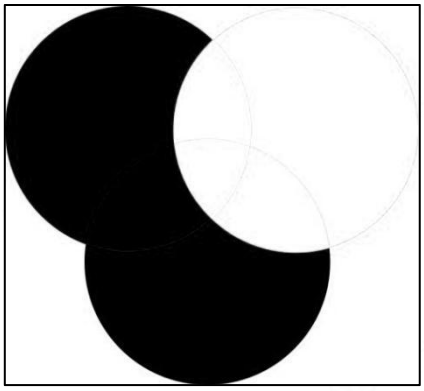

图 3-1-5 RGB 彩色图像的 G 通道

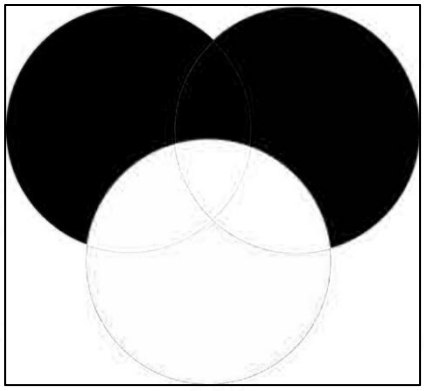

图 3-1-6 RGB 彩色图像的 G 通道

Lab 颜色空间，L 值范围[0, 128]代表亮度；a 值范围[-128, 127]，正数代表红色，负端代表绿色；b 值范围[-128, 127]，正数代表黄色，负端代表蓝色。

HSV 颜色空间，色调(H)，饱和度(S)，明度(V)在不同应用场景中，HSV 取值范围不尽相同，在此就不再展开了。

② 使用 OpenCV 进行图像转换与通道分离

接下来运行我们的例程程序进行 OpenCV 的图像转换与通道分离。

输入命令：python 1.Imageformat.py，即可执行，必须保证我们的例程图片【RGB.jpg】与 py 文件【1.Imageformat.py】位于同一个文件夹下。

下面是使用 OpenCV 对图像进行图像转换和通道分离程序。程序执行流程大致如下图 2-1-7 所示。

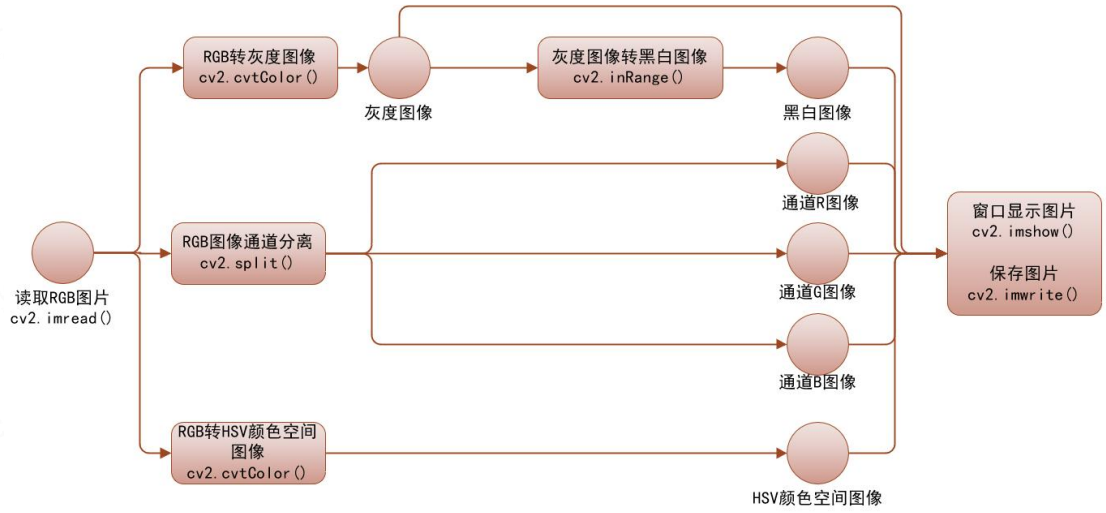

图 3-1-7 程序例程执行流程图

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
```

```
#1:为了防止用户没有将python安装在默认的/usr/bin目录，系统会先从env(系统环境变量)里查找python的安装路径，再调用对应路径下的解析器完成操作，也可以指定python3
#2:Python.X 源码文件默认使用utf-8编码，可以正常解析中文，一般而言，都会声明为utf-8编码

import cv2 #引用opencv功能包

#创建窗口，用于显示图片
cv2.namedWindow('MyWindow', cv2.WINDOW_NORMAL)

#提示停止方法
print ('Press key "Q" to stop.')

frame = cv2.imread('RGB.jpg',1) #以彩色图像格式读取图片
Quit=0 #是否继续运行标志位

while Quit==0:
    keycode=cv2.waitKey(3) #每3ms刷新一次图片，同时读取3ms内键盘的输入
    if(keycode==ord('Q')):#如果按下“Q”键，停止运行标志位置1，调出while循环，程序停止运行
       Quit=1
    cv2.imshow('MyWindow', frame) #显示原图

    #RGB转灰度图像
    frame_gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY) #注意是BGRopencv读取图片的默认像素排列是BGR
    cv2.imshow('frame_gray', frame_gray) #直接imshow(),系统会自动执行nameWindow()创建窗口
    cv2.imwrite('gray.jpg',frame_gray) #保存灰度图像为文件

    #灰度图像二值化处理获得黑白图像
    frame_binary = cv2.inRange(frame_gray,100,150) #灰度值在100-150之间的像素点置为白色
    cv2.imshow('frame_binary', frame_binary) #显示二值化图像
    cv2.imwrite('frame_binary.jpg',frame_binary) #显示二值化图像

    #RGB通道分离
    b, g ,r =cv2.split(frame) #注意返回值顺序为b、g、r,因为opencv读取图片的默认像素排列是BGR
    #显示三通道图像
    cv2.imshow('r', r)#二值化后的通道R
    cv2.imshow('g', g)#二值化后的通道G
    cv2.imshow('b', b)#二值化后的通道B
    #分别保存三通道图片
```

```
    cv2.imwrite('r.jpg',r)
    cv2.imwrite('g.jpg',g)
    cv2.imwrite('b.jpg',b)

    #BGR 转 HSV 颜色空间
    frame_HSV = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    cv2.imshow('frame_HSV', frame_HSV) #显示图片
    cv2.imwrite('frame_HSV.jpg',frame_HSV) #保存图片

    frame = cv2.imread('RGB.jpg',1) #循环读取图片

print ('Quitted!') #提示程序已停止
cv2.destroyAllWindows() #程序停止前关闭所有窗口
```

3.2 阈值分割(图像二值化)

① 概述

阈值分割是一种图像分割技术，又称为阈值二值化。

阈值分割在灰度图像最常用的方法是取一个阈值分界值 Threshold，然后当任意一个像素点的灰度值高于 Threshold 时，该像素点的灰度值直接取值为 255(该值可变)，当任意一个像素点的灰度值低于 Threshold 时，该像素点的灰度值直接直接取值为 0。即

$$dst(x,\ y) = \begin{cases} 255,\ dst(x,\ y) > \text{Threshold} \\ 0,\ dst(x,\ y) > \text{Threshold} \end{cases}$$

如下图所示，右图 2-2-2 黑白图像为 Threshold=128 时灰度图像的二值化结果。二值化的主要作用是降低数据处理复杂程度，因为将 8 位(256)数据降低到了 1 位(0/1)数据，另一个作用则是高亮标示目标。

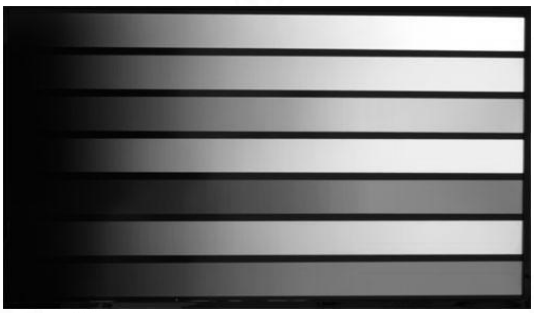

图 3-2-1 灰度图像

图 3-2-2 黑白图像

OpenCV_Python 的二值化 API：

Binary = cv2.threshold (Gray, 128, 255, cv2.THRESH_BINARY)

Gray 代表输入的灰度图像，128 为阈值，255 代表灰度值大于阈值的像素点的灰度值会被设置为 255，cv2.THRESH_BINARY 为二值化方法。

注意：该函数处理后图像依然为灰度图像，只是图像灰度值只有两个，需要手动转为二值(黑白)图像。

上面讲的是取一个阈值分界值 Threshold 进行二值化，这种方法有一个缺点就是只能将图像分割成两块：[0, Threshold]、[Threshold, 255]。很多时候我们的模板只是属于一个阈值范围[Threshold_Min, Threshold_Max]。公式如下

$$dst(x,\ y) = \begin{cases} 255,\ dst(x,\ y) \in \text{Threshold}[\text{Min},\ \text{Max}] \\ 0,\ Otherwise \end{cases}$$

如下图 2-2-4 所示，右图黑白图像为 Threshold=[100, 150] 时灰度图像的二值化结果。

图 3-2-3 灰度图像

图 3-2-4 黑白图像

② **彩色图像的二值化**

彩色图像的二值化的原理与灰度图像差不多，以 RGB 图像为例，只需要分别将 R、G、B 三个通道分离出来，然后对三通道的灰度图像分别进行二值化处理得到三通道的二值图像，最后将 R、G、B 三个通道的二值图像进行相与处理即可。公式如下。

对三通道的灰度图像分别进行二值化处理：

$$dst(x,\ y).\mathrm{R} = \begin{cases} 255,\ dst(x,\ y).\mathrm{R} \in [R_{min},\ R_{max}] \\ 0,\ Otherwise \end{cases}$$

$$dst(x,\ y).\mathrm{G} = \begin{cases} 255,\ dst(x,\ y).\mathrm{G} \in [G_{min},\ G_{max}] \\ 0,\ Otherwise \end{cases}$$

$$dst(x,\ y).\mathrm{B}=\begin{cases}255, & dst(x,\ y).\mathrm{G}\in[B_{min},\ B_{max}]\\ 0, & Otherwise\end{cases}$$

对三通道的灰度图像分别进行二值化处理：

$$dst(x,\ y).\mathrm{RGB}=dst(x,\ y).\mathrm{R}\ \&\&dst(x,\ y).\mathrm{G}\ \&\&dst(x,\ y).\mathrm{B}$$

下图为彩色图像在[R_{min},R_{max},G_{min},G_{max},B_{min},B_{max}]=[145, 255, 0, 74 ,0, 70]时的二值化结果。

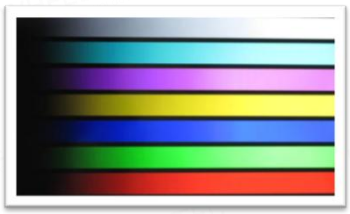

图 3-2-5 彩色图像

图 3-2-6 通道 R 二值化结果

图 3-2-7 通道 G 二值化结果

图 3-2-8 通道 B 二值化结果

图 3-2-9 三通道二值化结果相与

其代码实现有两个方法。一个就是如上过程：通道分离，3 通道分别二值化，3 通道二值化结果相与：

```
b, g ,r =cv2.split(Image) #通道分离
Binary_B = cv2.inRange(b, B_min, B_max)
Binary_G = cv2.inRange(g, G_min, G_max)
Binary_R = cv2.inRange(r, R_min, R_max)
Binary_RGB_AND = cv2.bitwise_and(Binary_R, Binary_G) #相与
Binary = cv2.bitwise_and(Binary_RGB_AND, Binary_B)#最终结果
注意：顺序为 BGR
```

另一个则是使用数组直接进行彩色图像二值化，该方法更加方便快捷，介绍第一种方法主要是想让大家熟悉彩色图像二值化的过程：

```
lower_bgr = np.array([B_min, G_min, R_min])
upper_bgr = np.array([B_max, G_max, R_max])
Binary = cv2.inRange(Image, lower_bgr, upper_bgr)#最终结果
注意：顺序为 BGR
```

③ 使用 OpenCV 对调用摄像头并进行动态阈值分割

接下来我们运行我们的例程程序，直观理解阈值分割的过程。本节例程程序除了阈值分割，还有两个重点，一个是调用摄像头，另一个是使用【Trackbar】进行动态调阈值。程序文件为【2.DynamicThreshold.py】。

程序运行效果如下图 2-2-10 所示，总共显示 10 个窗口，分别是原图窗口、动态调阈值窗口、原图 3 通道 3 个窗口、原图 3 通道二值化 3 个窗口、手动二值化图像窗口、数组二值化图像窗口。

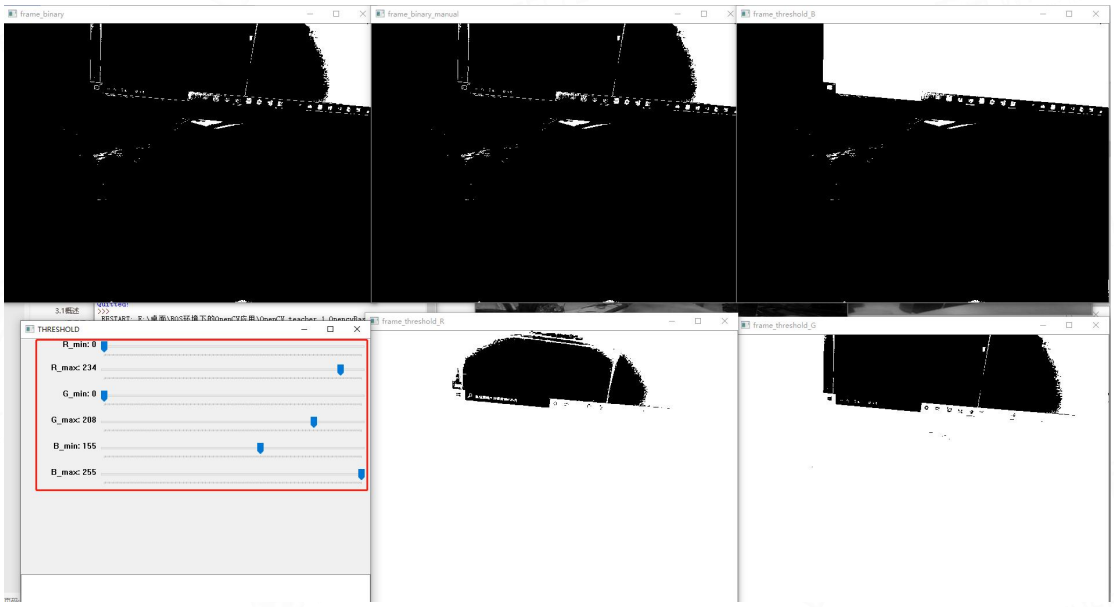

图 3-2-10 例程程序运行效果图

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为 utf-8 编码

import cv2 #引用 opencv 功能包
import numpy as np #引用数组功能包
```

```
#调阈值回调函数
def callback(object):
    pass

#创建窗口
cv2.namedWindow('MyWindow')

#提示停止方法
print ('Showing camera. Press key "Q" to stop.')

cameraCapture = cv2.VideoCapture(0) #创建读取摄像头的类
succes, frame = cameraCapture.read() #读取第一帧图片，返回值为(是否成功读取， 图片)

#创建画布、窗口、进度条
canvas = np.zeROS2((170, 600, 3), dtype=np.uint8) + 255 #创建画布放置阈值动态调节窗口
cv2.imshow("THRESHOLD",canvas)
cv2.createTrackbar("R_min","THRESHOLD",0,255,callback) #输入参数(参数名字，进度条附着窗口名字，进度条最小值，进度条最大值，回调函数)
cv2.createTrackbar("R_max","THRESHOLD",0,255,callback)
cv2.createTrackbar("G_min","THRESHOLD",0,255,callback)
cv2.createTrackbar("G_max","THRESHOLD",0,255,callback)
cv2.createTrackbar("B_min","THRESHOLD",0,255,callback)
cv2.createTrackbar("B_max","THRESHOLD",0,255,callback)

Quit=0 #是否继续运行标志位
while succes  and not Quit:
    keycode=cv2.waitKey(1)
    if(keycode==ord('Q')): #如果按下“Q”键，停止运行标志位置 1，调出 while 循环，程序停止运行
       Quit=1

    #相关变量绑定进度条
    R_min = cv2.getTrackbarPos("R_min","THRESHOLD",) #获得进度条值
    G_min = cv2.getTrackbarPos("G_min","THRESHOLD",)
    B_min = cv2.getTrackbarPos("B_min","THRESHOLD",)
    R_max = cv2.getTrackbarPos("R_max","THRESHOLD",)
    G_max = cv2.getTrackbarPos("G_max","THRESHOLD",)
    B_max = cv2.getTrackbarPos("B_max","THRESHOLD",)

    #分别对 RGB 三通道进行二值化
    b, g ,r =cv2.split(frame)       #RGB 通道分离
```

```
    cv2.imshow('r', r)#通道 R
    cv2.imshow('g', g)#通道 G
    cv2.imshow('b', b)#通道 B
    frame_threshold_B = cv2.inRange(b,B_min,B_max) #通道 R 二值化
    frame_threshold_G = cv2.inRange(g,G_min,G_max) #通道 G 二值化
    frame_threshold_R = cv2.inRange(r,R_min,R_max) #通道 B 二值化
    cv2.imshow('frame_threshold_R', frame_threshold_R)#窗口显示二值化后的通道 R
    cv2.imshow('frame_threshold_G', frame_threshold_G)#窗口显示二值化后的通道 G
    cv2.imshow('frame_threshold_B', frame_threshold_B)#窗口显示二值化后的通道 B

    #3 通道二值化结果相与
    Binary_RGB_AND = cv2.bitwise_and(frame_threshold_R, frame_threshold_G) #相与
    frame_binary_manual = cv2.bitwise_and(Binary_RGB_AND, frame_threshold_B)#最
终结果
    cv2.imshow('frame_binary_manual', frame_binary_manual) #窗口显示彩色图像手动
二值化结果

    #直接对 RGB 图像进行二值化
    lower_rgb = np.array([B_min, G_min, R_min]) #注意这里是 BGR,
    upper_rgb = np.array([B_max, G_max, R_max]) #因为 opencv 读取图片的默认像素排
列是 BGR
    frame_binary = cv2.inRange(frame,lower_rgb,upper_rgb) #使用数组进行图像二值
化

    cv2.imshow('MyWindow', frame) #窗口显示原图
    cv2.imshow('frame_binary', frame_binary)#窗口显示对 RGB 图像进行二值化的结果

    succes, frame = cameraCapture.read() #循环读取摄像头

if succes==0: #提示由于摄像头读取失败停止程序
    print ('Camera disconnect !')
print ('Quitted!') #提示程序已停止
cv2.destroyAllWindows() #程序停止前关闭所有窗口
cameraCapture.release #程序停止前关闭摄像头调用
```

3.3 膨胀腐蚀

① 概述

在图像处理中膨胀腐蚀是最基本也是最常用的操作，它常用来清除图像中的杂质，在目标识别时也常来构建连通域以便进行抠图。

如果把二值图像当做一个值只有 0 和 1 的矩阵，1 代表白色，0 代表黑色。

膨胀就是每个元素与周围一定范围内的元素进行或操作，与原值相比变化了

则保持变化，结果就是白色部分膨胀(黑色部分变小)了一定范围。

腐蚀就是每个元素与周围一定范围内的元素进行与操作，与原值相比变化了则保持变化，结果就是白色部分被腐蚀(黑色部分变大)了一定范围。

② 实例展示

下图为一张彩色图像，进行二值化后，再进行膨胀腐蚀清除杂质的过程。

先膨胀，清除白色矩形内的杂质，由于进行了膨胀，图像发生了变形(白色部分变大)，我们再进行腐蚀，使图像返回原来的形状。这时我们发现白色矩形内的黑斑没有出现。

然后我们再进行一次腐蚀然后进行膨胀，把左上角的白色杂质也清除掉。

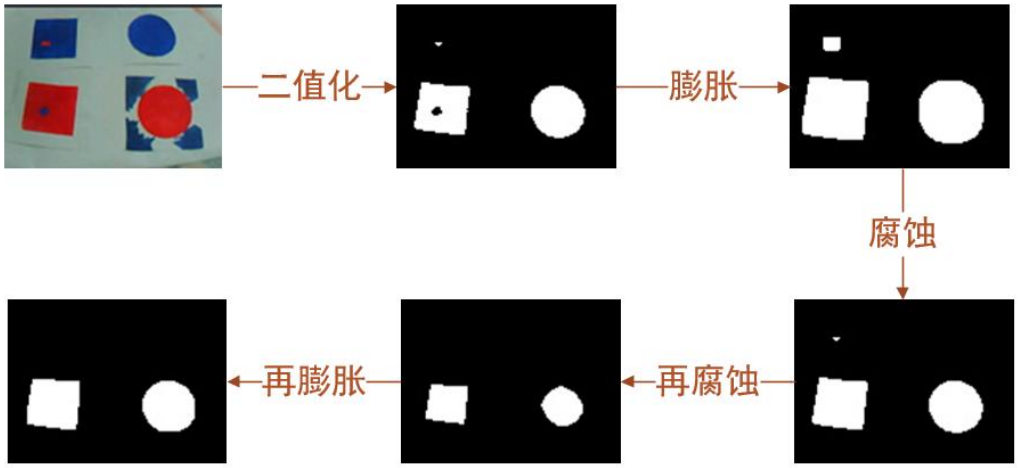

图 3-3-1 膨胀腐蚀实例

下面为对二值化图像进行膨胀腐蚀的程序实现。

```
kernel_width=7  #膨胀腐蚀的范围大小
kernel_height=7 #膨胀腐蚀的范围大小
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_width, kernel_height))
#创建膨胀腐蚀核
frame_threshold_D         = cv2.dilate(frame_threshold, kernel)       #膨胀
frame_threshold_D_E       = cv2.erode(frame_threshold_D, kernel)      #腐蚀
frame_threshold_D_E_E     = cv2.erode(frame_threshold_D_E, kernel)    #腐蚀
frame_threshold_D_E_E_D   = cv2.dilate(frame_threshold_D_E_E, kernel) #膨胀
```

膨胀腐蚀是图像处理中很重要的一个操作，但是理解起来也并不难，使用过程也并不复杂。所以本节就不给出完整程序在文档里了，但是我们本节还是提供了程序，文件为【3.DynamicDilateErode.py】。在上一节的基础上添加了动态调膨胀腐蚀核大小的功能。运行效果为，多了个膨胀腐蚀结果窗口，和动态调阈值窗口多了两个进度条：膨胀腐蚀和的高度和宽度。

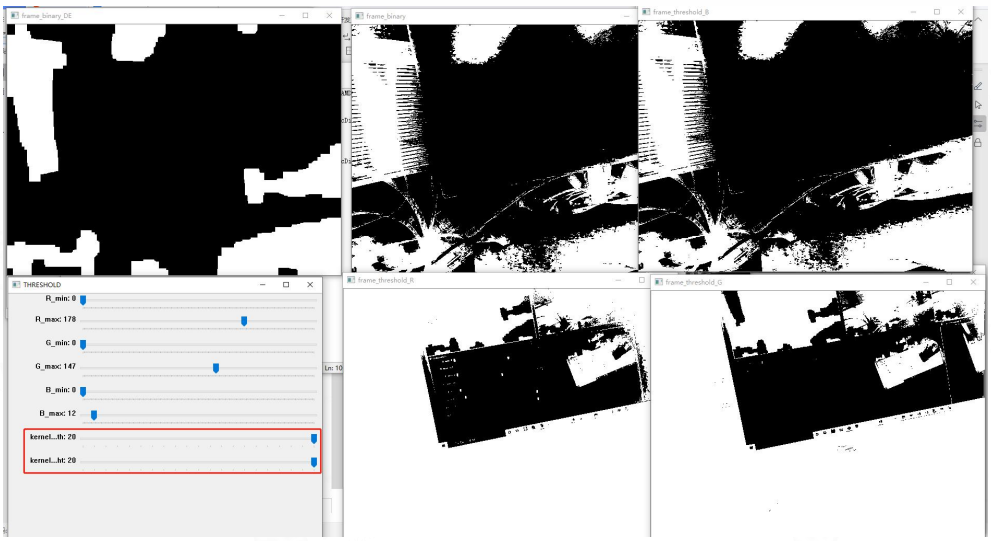

图 3-2-2 例程程序运行效果图

4. OpenCV 应用例程

4. 1 自动提取目标阈值

① 概述

本节例程主要用到直方图信息处理的知识。

其程序执行流程为：前 3 秒时间给使用者将摄像头对准目标，第 3 秒到第 13 秒程序通过处理直方图信息提取目标的阈值，13 秒后使用阈值结果进行二值化并显示效果，同时会将阈值结果打印出来。下图 4-1-1 为程序正在提取阈值，图 4-1-2 为自动提取阈值的二值化效果。

程序文件为【1.AutoAttractThreshold.py】。

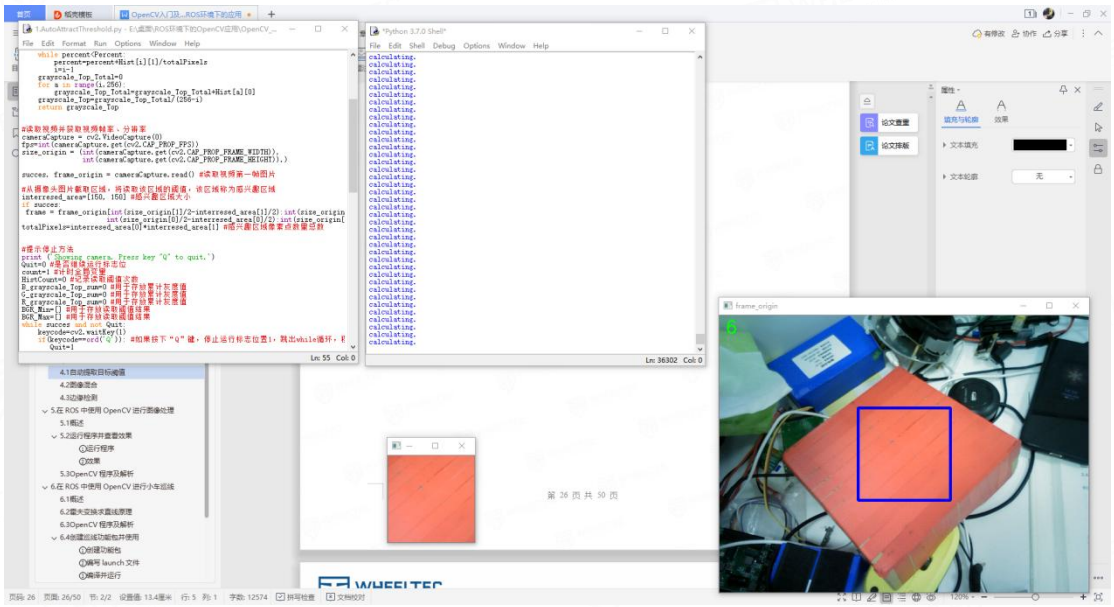

图 4-1-1 程序正在提取阈值

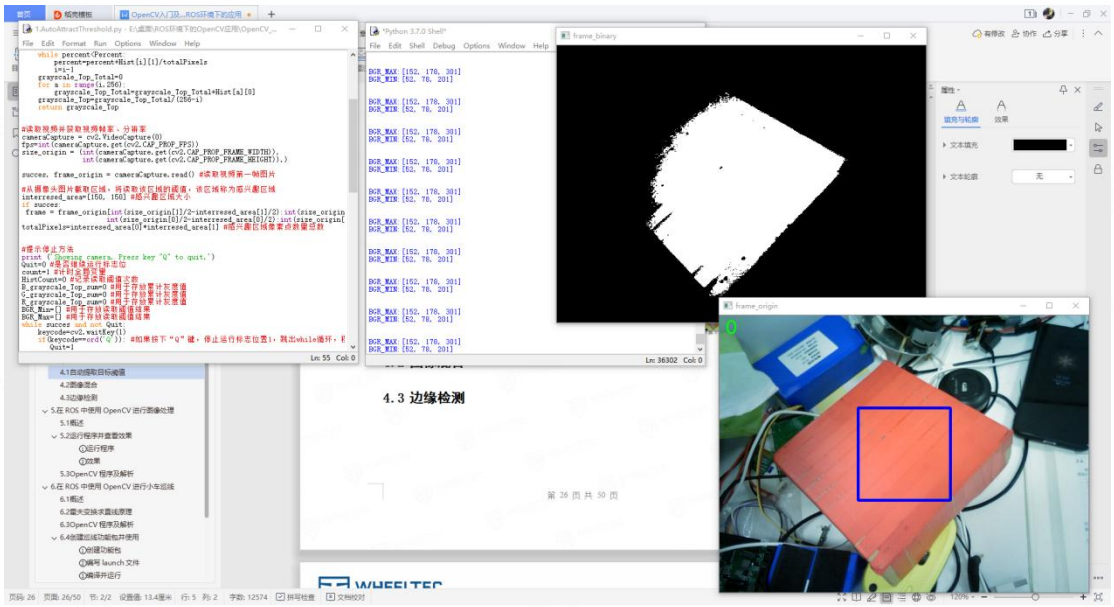

图 4-1-2 自动提取阈值的二值化效果

② 程序与解析

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为 utf-8 编码

import cv2 #引用 OpenCV 功能包
import numpy as np #引用数组功能包

#输入直方图数据，并输出新的直方图数据。新直方图数据按照像素点个数重新排序，数组元素为二元数组[灰度值，像素点个数]
def ResortHist(hist):
    Hist=[]
    i=0
    for pixle in hist:
        a=[i, int(pixle[0])]
        i=i+1
        Hist.append(a)
    Hist.sort(key = lambda x : (x[1]))#按照第二个元素进行升序排序
    return Hist

#输入 2 元直方图数据、百分比，输出占比前百分比的灰度值平均值
def getTopPercentGrayscale(Hist, Percent):
    #求 B 通道占比前 99%的像素点的灰度值平均值
```

```
    percent=0
    i=len(Hist)-1
    while percent<Percent/100 and i>=0:
        percent=percent+Hist[i][1]/totalPixels
        i=i-1
    grayscale_Top_Total=0
    for a in range(i,256):
        grayscale_Top_Total=grayscale_Top_Total+Hist[a][0]
    grayscale_Top=grayscale_Top_Total/(256-i)
    return grayscale_Top


#读取视频并获取视频帧率、分辨率
cameraCapture = cv2.VideoCapture(0)
fps=int(cameraCapture.get(cv2.CAP_PROP_FPS))
size_origin = (int(cameraCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
               int(cameraCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)),)

succes, frame_origin = cameraCapture.read() #读取视频第一帧图片

#从摄像头图片截取区域，将读取该区域的阈值，该区域称为感兴趣区域
interresed_area=[150, 150] #感兴趣区域大小
if succes:
 frame =
frame_origin[int(size_origin[1]/2-interresed_area[1]/2):int(size_origin[1]/2+in
terresed_area[1]/2),

int(size_origin[0]/2-interresed_area[0]/2):int(size_origin[0]/2+interresed_area
[1]/2)]
totalPixels=interresed_area[0]*interresed_area[1] #感兴趣区域像素点数量总数


#提示停止方法
print ('Showing camera. Press key "Q" to quit.')
Quit=0 #是否继续运行标志位
count=1 #计时全局变量
HistCount=0 #记录读取阈值次数
B_grayscale_Top_sum=0 #用于存放累计灰度值
G_grayscale_Top_sum=0 #用于存放累计灰度值
R_grayscale_Top_sum=0 #用于存放累计灰度值
BGR_Min=[] #用于存放读取阈值结果
BGR_Max=[] #用于存放读取阈值结果
while succes and not Quit:
    keycode=cv2.waitKey(1)
```

```
    if(keycode==ord('Q')): #如果按下“Q”键，停止运行标志位置 1，跳出 while 循环，
程序停止运行
        Quit=1

    #等待 3 秒后开始读取阈值，读取 10 秒(根据摄像头帧率 30 计算时间)
    if(count>90 and count<390):
        #获得感兴趣区域直方图数据
        hist_B = cv2.calcHist([frame], [0], None, [256], [0,256])
        hist_G = cv2.calcHist([frame], [1], None, [256], [0,256])
        hist_R = cv2.calcHist([frame], [2], None, [256], [0,256])

        #创建新的 BGR 直方图数组，数组元素为二元数组[灰度值，像素点个数]
        Hist_B=ReSortHist(hist_B)
        Hist_G=ReSortHist(hist_G)
        Hist_R=ReSortHist(hist_R)

        #求 BGR 通道占比前 45%的像素点的灰度值平均值
        B_grayscale_Top_temp=getTopPercentGrayscale(Hist_B,45)
        G_grayscale_Top_temp=getTopPercentGrayscale(Hist_G,45)
        R_grayscale_Top_temp=getTopPercentGrayscale(Hist_R,45)

        #累计
        B_grayscale_Top_sum=B_grayscale_Top_sum+B_grayscale_Top_temp
        G_grayscale_Top_sum=G_grayscale_Top_sum+G_grayscale_Top_temp
        R_grayscale_Top_sum=R_grayscale_Top_sum+R_grayscale_Top_temp
        HistCount=HistCount+1

        #循环显示与截取感兴趣区域
        cv2.imshow("frame", frame)
        frame =
frame_origin[int(size_origin[1]/2-interresed_area[1]/2):int(size_origin[1]/2+in
terresed_area[1]/2),

int(size_origin[0]/2-interresed_area[0]/2):int(size_origin[0]/2+interresed_area
[0]/2)]
        print("calculating.")


    #读取完毕，计算阈值
    if(count==390):
        cv2.destroyWindow("frame") #关闭感兴趣区域显示窗口
        B_grayscale_Top=B_grayscale_Top_sum/HistCount
        G_grayscale_Top=G_grayscale_Top_sum/HistCount
        R_grayscale_Top=R_grayscale_Top_sum/HistCount
```

```
        ThresholdTolerance=50
        BGR_Min=[int(B_grayscale_Top-ThresholdTolerance),
                 int(G_grayscale_Top-ThresholdTolerance),
                 int(R_grayscale_Top-ThresholdTolerance)]
        BGR_Max=[int(B_grayscale_Top+ThresholdTolerance),
                 int(G_grayscale_Top+ThresholdTolerance),
                 int(R_grayscale_Top+ThresholdTolerance)]

    #计算完毕，打印阈值,并显示该阈值的二值化效果
    if(count>390):
        print("BGR_MAX:"+str(BGR_Max))
        print("BGR_MIN:"+str(BGR_Min))
        print("\n\t")
        frame_binary =
cv2.inRange(frame_origin,np.array(BGR_Min),np.array(BGR_Max))
        cv2.imshow("frame_binary", frame_binary)

    #在摄像头原图显示感兴趣区域并显示计时
    cv2.rectangle(frame_origin, (int(size_origin[0]/2-interresed_area[0]/2-5),
int(size_origin[1]/2-interresed_area[1]/2)-5),
                                (int(size_origin[0]/2+interresed_area[0]/2+5),
int(size_origin[1]/2+interresed_area[1]/2)+5), (255, 0, 0), 3)
    timeShow=int(13-count/30)
    if timeShow<0:timeShow=0
    cv2.putText(frame_origin, str(timeShow), (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
1, (0, 255, 0), 2)

    #循环显示与读取摄像头
    cv2.imshow("frame_origin", frame_origin)
    succes, frame_origin = cameraCapture.read() #读取视频第一帧
    count=count+1

if succes==0: #提示由于摄像头读取失败停止程序
    print ('Camera disconnect !')
print ('Quitted!') #提示程序已停止
cameraCapture.release() #释放摄像头
cv2.destroyAllWindows() #程序停止前关闭所有窗口
```

4.2 图像混合

① 概述

本节将使用像素点操作对两张不同尺寸的图片进行融合并居中显示。

下图 4-2-1 为程序运行效果。程序文件为【2.ImageMerge.py】。

图 4-2-1 图片融合程序运行效果

② 程序与解析

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为 utf-8 编码

import cv2 #引用 OpenCV 功能包
import numpy as np #引用数组功能包

#读取图片
img_1 = cv2.imread('2.ImageMerge_1.jpg')
img_2 = cv2.imread('2.ImageMerge_2.jpg')

#获取图片信息
img_1_height, img_1_width, img_1_channels = img_1.shape
img_2_height, img_2_width, img_2_channels = img_2.shape

#设定融合图片的高度、宽度
img_merged_height=img_1_height
img_merged_width=img_1_width
if(img_2_height>img_1_height):img_merged_height=img_2_height
if(img_2_width>img_1_width):  img_merged_width=img_2_width

#创建融合图片对象
```

```
img_merged = np.zeROS2((img_merged_height, img_merged_width, 3), dtype=np.uint8)

#将 img_1 赋值在融合图片大小的画幅中并居中显示保存为 img_1_new
img_1_new=np.zeROS2((img_merged_height, img_merged_width, 3), dtype=np.uint8)
x_bias_1=int((img_merged_width-img_1_width)/2)
y_bias_1=int((img_merged_height-img_1_height)/2)
for col in range(img_1_width):
     for row in range(img_1_height):
          img_1_new[row+y_bias_1, col+x_bias_1]=img_1[row, col]

#将 img_2 赋值在融合图片大小的画幅中并居中显示保存为 img_2_new
img_2_new=np.zeROS2((img_merged_height, img_merged_width, 3), dtype=np.uint8)
x_bias_2=int((img_merged_width-img_2_width)/2)
y_bias_2=int((img_merged_height-img_2_height)/2)
for col in range(img_2_width):
     for row in range(img_2_height):
          img_2_new[row+y_bias_2, col+x_bias_2]=img_2[row, col]

#将 img_1_new 与 img_2_new 融合并赋值给 img_merged
for col in range(img_merged_width):
     for row in range(img_merged_height):
               img_merged[row, col]= (img_1_new[row, col]/2+img_2_new[row, col]/2)

Quit=0 #是否继续运行标志位
#提示停止方法
print ('Press key "Q" to stop.')

while Quit==0:
     keycode=cv2.waitKey(3) #每 3ms 刷新一次图片，同时读取 3ms 内键盘的输入
     if(keycode==ord('Q')):#如果按下“Q”键，停止运行标志位置 1，调出 while 循环，
程序停止运行
        Quit=1
     #窗口显示图片
     cv2.imshow("img_1", img_1)
     cv2.imshow("img_2", img_2)
     cv2.imshow("img_1_new", img_1_new)
     cv2.imshow("img_2_new", img_2_new)
     cv2.imshow("img_merged", img_merged)

print ('Quitted!') #提示程序已停止
cv2.destroyAllWindows() #程序停止前关闭所有窗口

cv2.imwrite('2.ImageMerge_merged.jpg', img_merged) #保存融合图片
```

4.3 自定义内核进行边缘检测

① 概述

本节使用 OpenCV 提供的卷积 API：cv2.filter2D()进行边缘检测。

本节提供了 3 个边缘检测算子，分别是横向边缘检测算子、垂直边缘检测算子和全向边缘检测算子。

程序运行效果如下图 4-3-1 所示。程序文件为【3.EdgeDetection.py】。

下面以横向边缘检测算子为例介绍一下图像处理中的卷积操作，同时设有一个 5*5 分辨率的灰度图片，其每个像素点的灰度值如图 4-3-2。

```
[[-3, -9, -3],
 [ 0,  0,  0],
 [ 3,  9,  3]]
```

图 4-3-1 横向边缘检测算子

```
[[1,  2,  3,  4, 5],
 [1,  2,  3,  4, 5],
 [1,  2,  3,  4, 5],
 [1,  2,  3,  4, 5],
 [10, 20, 30, 4, 5] ]
```

图 4-4-2 5*5 分辨率的图片

对图片进行卷积即使图片的每个像素点与算子进行卷积。

以第 2 行第 2 列的像素点为例，其进行卷积后的灰度值为|(-3*1)+(-9*2)+(-3*3)+(0*1)+(0*2)+(0*3)+(3*1)+(9*2)+(3*3)|=0。即“像素点与算子大小相当的范围内的灰度值”与“算子的对应位置的值”相乘再相加。

以第 4 行第 2 列的像素点为例，其进行卷积后的灰度值为|(-3*1)+(-9*2)+(-3*3)+(0*1)+(0*2)+(0*3)+(3*10)+(9*20)+(3*30)|=270(注意有绝对值符号)。270 大于灰度值上限 255，取值 255。

可以发现，当像素点上下一行的灰度值相差较大时，其卷积后的灰度值也更大，而当两行的灰度值差别很大则是在边缘才会出现的情况。上下两行的灰度值差别很大代表横向边缘，左右两行的灰度值差别很大代表垂直边缘。

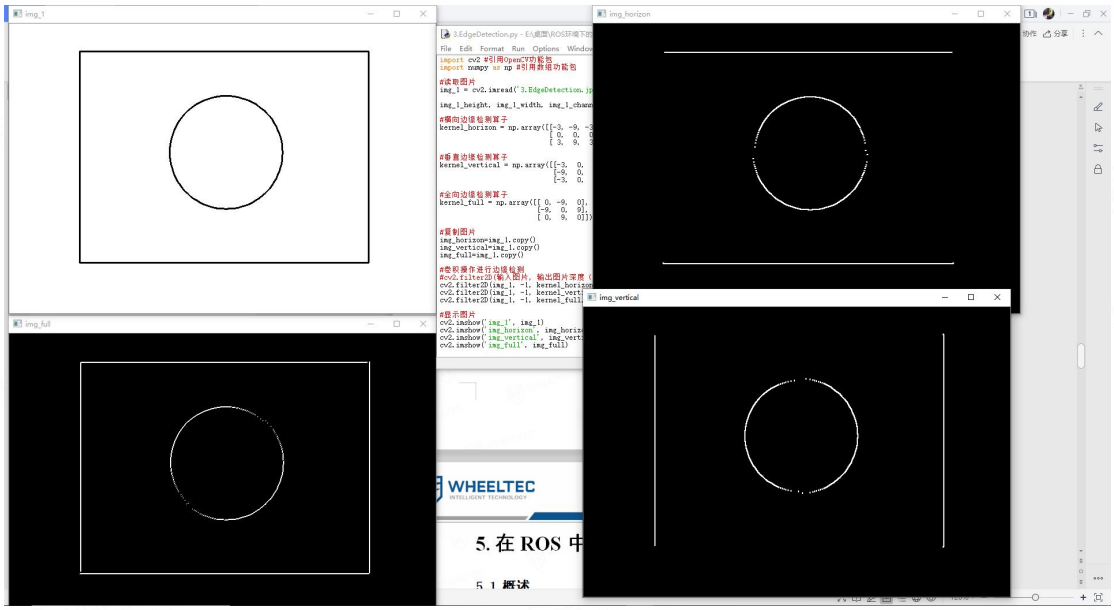

图 4-5-3 程序运行效果图

② 程序与解析

```
#!/usr/bin/env python
# coding=utf-8
#1.编译器声明和 2.编码格式声明
#1:为了防止用户没有将 python 安装在默认的/usr/bin 目录，系统会先从 env(系统环境变量)里查找 python 的安装路径，再调用对应路径下的解析器完成操作，也可以指定 python3
#2:Python.X 源码文件默认使用 utf-8 编码，可以正常解析中文，一般而言，都会声明为 utf-8 编码

import cv2 #引用 OpenCV 功能包
import numpy as np #引用数组功能包

#读取图片
img_1 = cv2.imread('3.EdgeDetection.jpg')

#获取图片信息
img_1_height, img_1_width, img_1_channels = img_1.shape

#横向边缘检测算子
kernel_horizon = np.array([[-3, -9, -3],
                           [ 0,  0,  0],
                           [ 3,  9,  3]])

#垂直边缘检测算子
kernel_vertical = np.array([[-3,  0,  3],
                            [-9,  0,  9],
                            [-3,  0,  3]])
```

```
#全向边缘检测算子
kernel_full = np.array([[ 0, -9,  0],
                        [-9,  0,  9],
                        [ 0,  9,  0]])

#复制图片
img_horizon=img_1.copy()
img_vertical=img_1.copy()
img_full=img_1.copy()

#卷积操作进行边缘检测
#cv2.filter2D(输入图片, 输出图片深度 (-1 表示与输入图片深度一样), 卷积核,  输出图片)
cv2.filter2D(img_1, -1, kernel_horizon,  img_horizon)
cv2.filter2D(img_1, -1, kernel_vertical, img_vertical)
cv2.filter2D(img_1, -1, kernel_full,     img_full)

Quit=0 #是否继续运行标志位
#提示停止方法
print ('Press key "Q" to stop.')

while Quit==0:
    keycode=cv2.waitKey(3) #每 3ms 刷新一次图片, 同时读取 3ms 内键盘的输入
    if(keycode==ord('Q')): #如果按下“Q”键, 停止运行标志位置 1, 调出 while 循环, 程序停止运行
        Quit=1

    #显示图片
    cv2.imshow('img_1', img_1)
    cv2.imshow('img_horizon', img_horizon)
    cv2.imshow('img_vertical', img_vertical)
    cv2.imshow('img_full', img_full)

print ('Quitted!') #提示程序已停止
cv2.destroyAllWindows() #程序停止前关闭所有窗口
```

5. 在 ROS2 中使用 OpenCV 进行图像处理

5.1 概述

要进行图像处理，那么首先就要有有一个图像来源，一个图像处理过程和一个图像处理结果。那么在 ROS2 中，图像来源就可以是一个节点发布的话题，图像处理过程也可以是一个节点，图像处理节点订阅图像来源发布的话题，图像处理结果则是图像处理节点发布的话题。

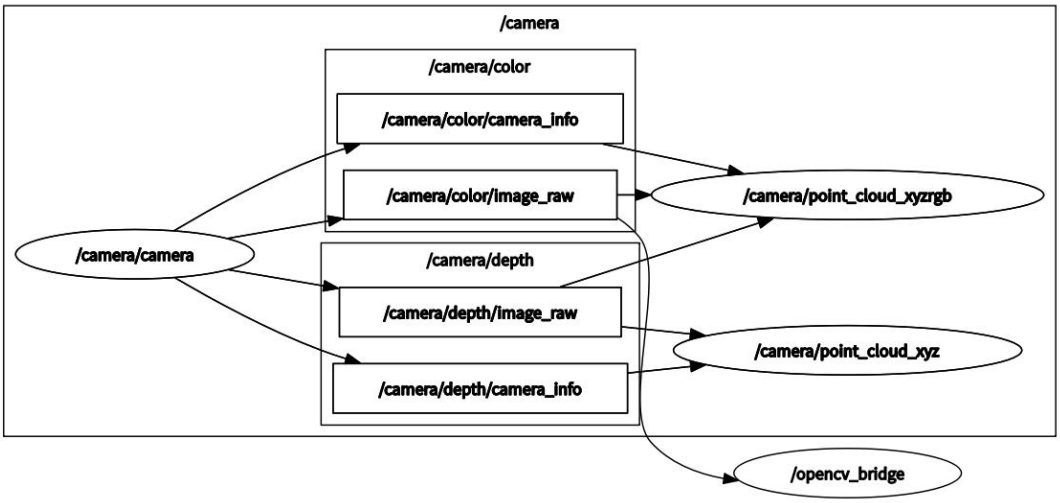

图 5-1-1 例程节点与话题通信架构

如上图 1-1 为本章例程的节点与话题的通信架构，其中椭圆形的代表节点，矩形的代表话题。可以看到节点【/camera/camera】发布了很多话题，其中包括【/camera/color/image_raw】，节点【/camera/camera】的作用主要是读取摄像头并发布相关图像话题。同时可以看到节点【/opencv_bridge】订阅了话题【/camera/color/image_raw】，本章的重点就是这个【/opencv_bridge】节点。

5.2 运行程序并查看效果

① 运行程序

ros2 launch turn_on_wheeltec_robot wheeltec_camera #启动【wheeltec_camera】节点

python3 opencv_ros2_test.py #运行程序启动【/opencv_bridge】节点

② 效果

运行 opencv_ros2_test.py 后，会弹出一个窗口【cv_image】,该窗口会显示当

前摄像头所拍摄到的画面，同时画面左上角会显示一大一小、一红一绿两个圆。

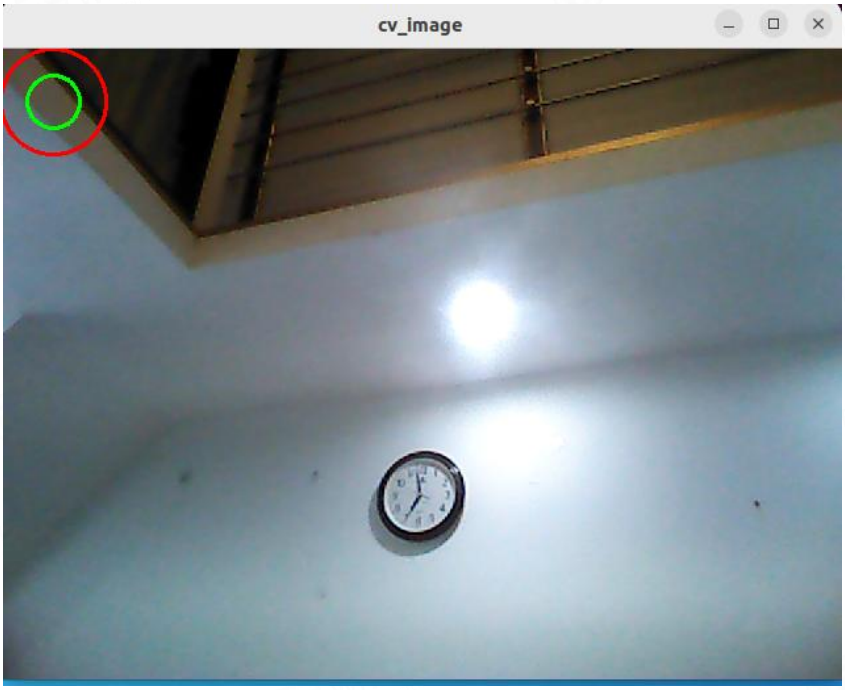

图 5-2-1 图片显示效果

打开新终端输入：rqt。会弹出一个窗口，点击窗口左上角下拉框，可以选择不同的图像话题，可以看到节点【/camera/camera】发布的话题【/camera/color/image_raw】和节点【/opencv_bridge】发布的话题【/cv_bridge_image】。

其中【/camera/color/image_raw】即为摄像头采集到的原始图像。【/cv_bridge_image】为经过 OpenCV 处理过的图像，它左上角有一个红色圆。而前面窗口【cv_image】显示的则是经过 OpenCV 二次处理过的图像。

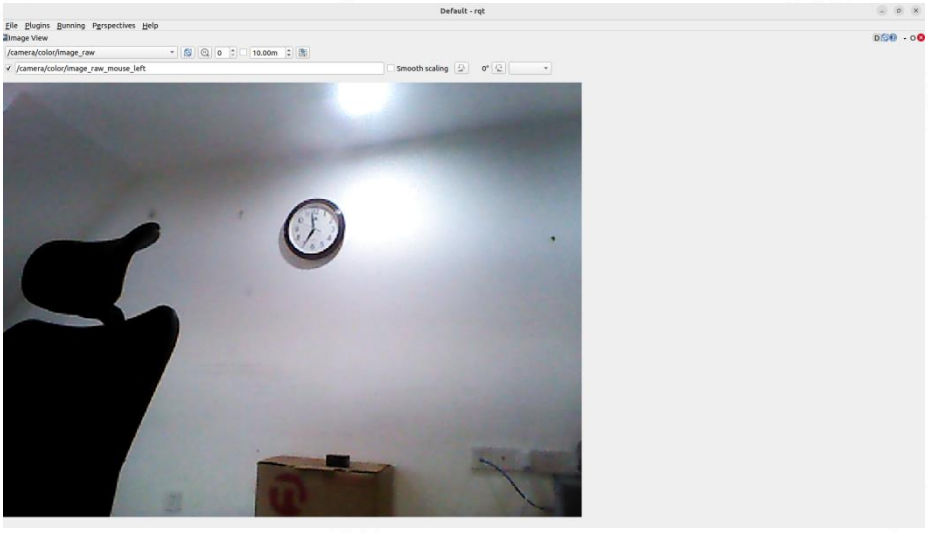

图 5-2-2 【/camera/color/image_raw】

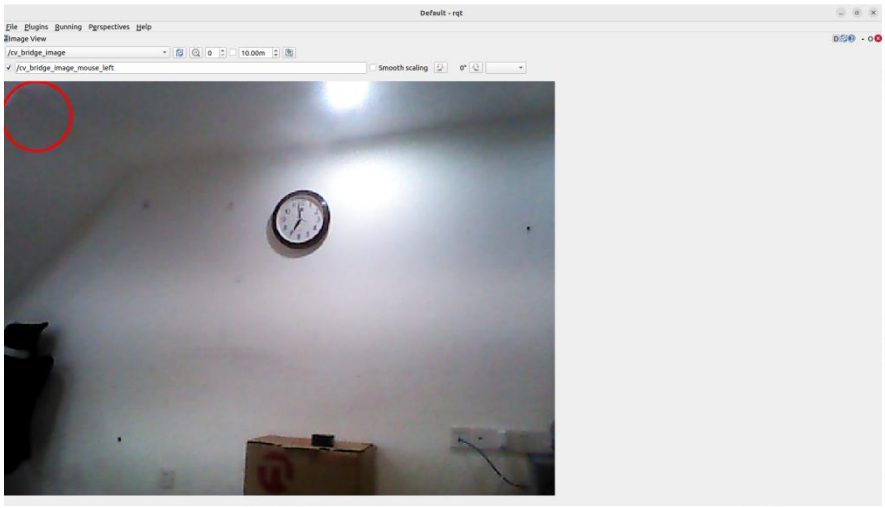

图 5-2-3 【/cv_bridge_image】

5.3 OpenCV 程序及解析

【/camera/camera】节点为读取摄像头并发布相关话题，该功能已集成，而且仅是读取并发布图像，并无太多图像处理内容。所以本节仅解析【opencv_ros2_test.py】的程序内容。该程序的执行过程大致如下图 1-3-1 所示。

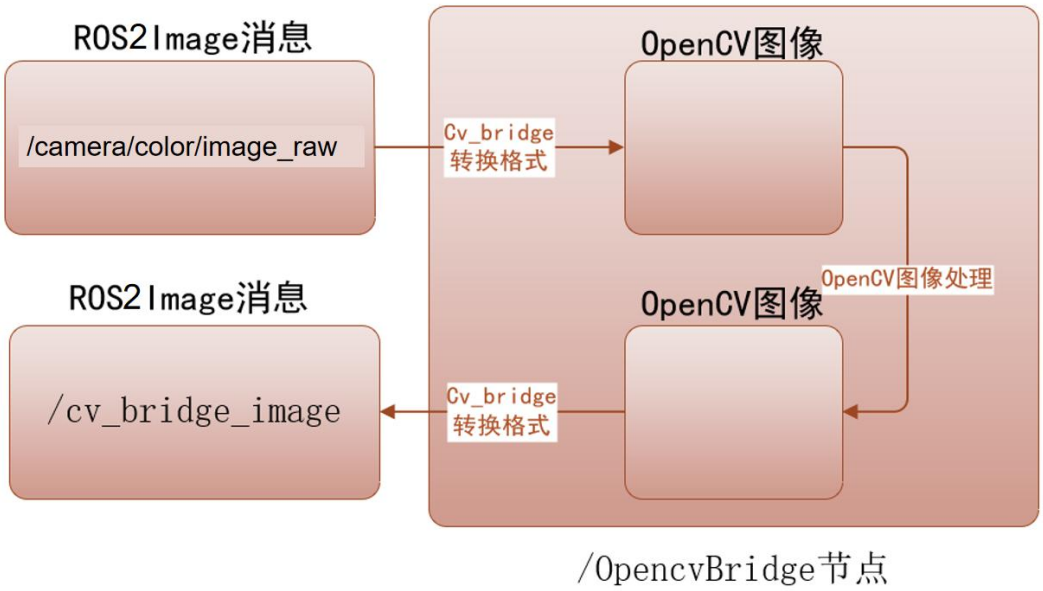

图 5-3-1 程序执行流程图

以下为【opencv_ros2_test.py】的源码与解析。

```
#!/usr/bin/env python3
# coding=utf-8

# 导入 ROS 2 相关库
import rclpy
```

```
from rclpy.node import Node
# 导入 cv_bridge 库，用于在 ROS 图像消息和 OpenCV 图像格式之间进行转换
from cv_bridge import CvBridge
# 导入 OpenCV 库，用于图像处理
import cv2
# 导入 ROS2 图像消息类型
from sensor_msgs.msg import Image


# 定义一个名为 ImageConverter 的类，继承自 rclpy.node.Node
# 功能：订阅 ROS2 图像消息，将其转换为 OpenCV 格式进行处理，处理完成后重新转换为
ROS 图像消息并发布
class ImageConverter(Node):
    def __init__(self):
        # 调用父类的初始化方法，将节点命名为'opencv_bridge'
        super().__init__('opencv_bridge')
        # 初始化 cv_bridge 对象，用于在 ROS2 图像消息和 OpenCV 图像格式之间进行转换
        self.bridge = CvBridge()
        # 创建一个订阅者，订阅'/usb_cam/image_raw'主题的图像消息
        # 消息类型为 sensor_msgs.msg.Image，回调函数为 self.callback，队列大小为
10
        self.subscription = self.create_subscription(
            Image,
            '/usb_cam/image_raw',
            self.callback,
            10)
        # 创建一个发布者，用于发布处理后的图像消息到'cv_bridge_image'主题
        self.publisher_ = self.create_publisher(Image, 'cv_bridge_image', 10)
        # 在节点的日志中输出提示信息，表示节点已启动
        self.get_logger().info('cv_bridge_test node started')

    def callback(self, msg):
        # 尝试将接收到的 ROS2 图像消息转换为 OpenCV 图像格式
        try:
            # 使用 cv_bridge 的 imgmsg_to_cv2 方法进行转换，指定目标编码为'bgr8'
            cv_image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
        except Exception as e:
            # 如果转换失败，记录错误信息并返回
            self.get_logger().error(f"Failed to convert image: {e}")
            return

        # 获取图像的行数、列数和通道数
        (rows, cols, channels) = cv_image.shape
        # 判断图像的宽度和高度是否大于 40 像素
```

```
        if cols > 40 and rows > 40:
            # 如果满足条件，在图像的(40, 40)位置绘制一个半径为 40 像素的红色圆圈
            # 参数分别为：图像、圆心坐标、半径、颜色(BGR)、线条粗细
            cv2.circle(cv_image, (40, 40), 40, (0, 0, 255), 2)

        # 尝试将处理后的 OpenCV 图像重新转换为 ROS 图像消息
        try:
            # 使用 cv_bridge 的 cv2_to_imgmsg 方法进行转换，指定编码为'bgr8'
            self.publisher_.publish(self.bridge.cv2_to_imgmsg(cv_image,
encoding='bgr8'))
        except Exception as e:
            # 如果转换失败，记录错误信息
            self.get_logger().error(f"Failed to convert image back: {e}")

        # 在图像的(40, 40)位置绘制一个半径为 20 像素的绿色圆圈（仅用于显示）
        cv2.circle(cv_image, (40, 40), 20, (0, 255, 0), 2)
        # 创建一个名为"cv_image"的窗口，窗口类型为可调整大小
        cv2.namedWindow("cv_image", cv2.WINDOW_NORMAL)
        # 在窗口中显示处理后的图像
        cv2.imshow("cv_image", cv_image)
        # 等待 10 毫秒，用于刷新窗口显示
        cv2.waitKey(10)


# 定义主函数，程序的入口
def main(args=None):
    # 初始化 ROS 2
    rclpy.init(args=args)
    # 创建 ImageConverter 节点实例
    image_converter = ImageConverter()
    # 保持节点运行，处理订阅的消息
    rclpy.spin(image_converter)
    # 销毁节点
    image_converter.destroy_node()
    # 关闭 ROS 2
    rclpy.shutdown()


# 判断是否为主程序运行，如果是，则调用主函数
if __name__ == '__main__':
    main()
```

6. 在 ROS2 中使用 OpenCV 进行小车巡线

6.1 概述

在进行了前面的基础学习后，再学习小车巡线就很简单了，本章将会用到前面学习到的所有内容，同时还有一个新内容：图像矩，本章巡线使用到的直线识别 API 就是基于图像矩原理的。其中的预设阈值和预设膨胀腐蚀核，则是由使用【Trackbar】动态调参得到，获得合适的阈值和膨胀腐蚀核后则关闭【Trackbar】节省系统资源以获得更好的巡线效果。

注意：本手册的例程是基于我们的 ROS2 机器人产品的，如果程序不是运行于我们的 ROS2 机器人产品上的话可能需要自行安装相关功能包。

6.2 图像矩求直线原理

图像矩是图像分析中的重要工具，用于描述形状的几何特征，在目标识别、形状匹配和姿态分析中有广泛应用。其核心思想是通过像素强度的加权积分计算特征量，不同阶次的矩可反映形状的面积、质心、方向等属性。

① 基本定义

对于二维图像函数 f(x,y)，其(p+q)阶矩定义为：

$$M_{pq} = \sum_{x}\sum_{y} x^p y^q f(x,y)$$

设二值图像中有一个 3×3 正方形区域，如下图所示，坐标范围为(1,1)到(3,3)，

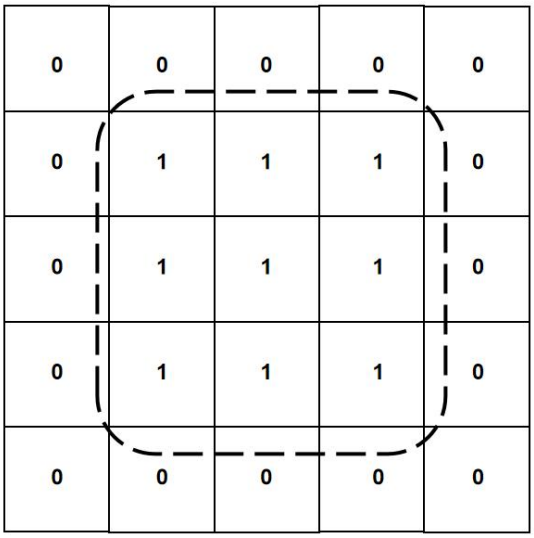

图 6-2-1 3×3 正方形区域

则：

- 零阶矩（面积）：M_{pq}=9（像素总数）
- 一阶矩：M_{pq}=(1+2+3)×3=18，M_{pq}=(1+2+3)×3=18
- 质心坐标：$\bar{x}$=M_{pq}/M_{pq}=2，$\bar{y}$=M_{pq}/M_{pq}=2

② 中心矩与归一化

为消除平移影响，定义中心矩：

$$\mu_{pq} = \sum_{x}\sum_{y}(x-\bar{x})^{p}(y-\bar{y})^{q}f(x,y)$$

对上例的正方形：

$$\mu_{20} = [(1-2)^2 + (2-2)^2 + (3-2)^2] \times 3 = 6$$

μ_{02}=6（对称性导致与μ_{20}相等）

μ_{11}=0（对称区域无相关性）

进一步通过归一化获得尺度不变性：

③ 主轴方向计算

利用二阶中心矩可计算形状方向：

$$\theta = \frac{1}{2}\arctan\left(\frac{2\mu_{11}}{\mu_{20}-\mu_{02}}\right)$$

若某长方形区域的μ_{20}=72，μ_{02}=36，μ_{11}=0，则：

$$\theta = \frac{1}{2}arctan\,(0) = 0°$$

表明该长方形的长轴与 x 轴平行

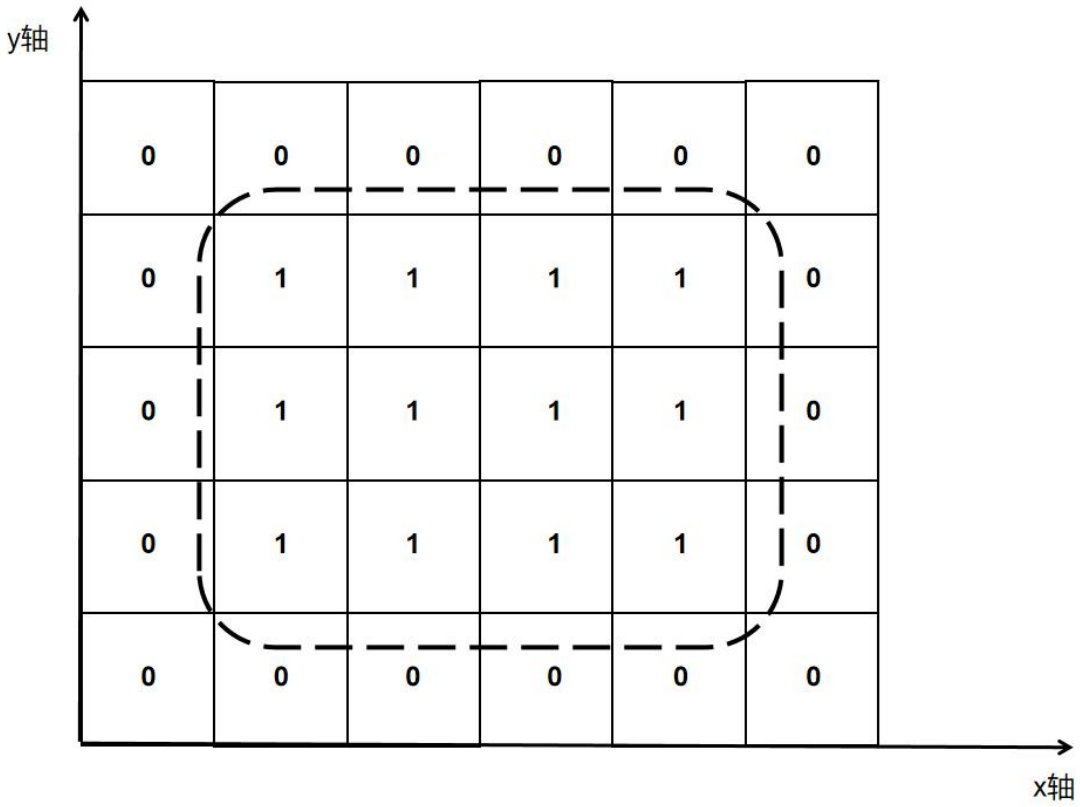

图 6-2-2　4×3 长方形区域

④　**Hu 不变矩**

通过非线性组合 7 个归一化中心矩，可得到具有旋转、平移、尺度不变性的 Hu 矩（OpenCV 中通过 moments()和 HuMoments()实现）：

$$h_1 = \eta_{20} + \eta_{02}$$
$$h_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$\vdots$$
$$h_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

6.3 OpenCV 程序及解析

程序文件名为【line_follow.py】。下面为源码及其解析。

```
#!/usr/bin/env python3

# 导入 OpenCV 库，用于图像处理
import cv2
# 导入 NumPy 库，用于数值计算
import numpy as np
# 导入 rclpy 库，用于 ROS 2 编程
import rclpy
```

```
# 导入 time 模块，用于时间相关操作
import time
# 导入 cv_bridge 库，用于在 ROS 图像消息和 OpenCV 图像格式之间进行转换
import cv_bridge
# 导入 ROS 图像消息类型
from sensor_msgs.msg import Image
# 导入 ROS 服务质量配置
from rclpy.qos import QoSProfile
# 导入 ROS Twist 消息类型，用于发布速度指令
from geometry_msgs.msg import Twist

# 全局变量，用于存储上一次的误差值
last_erro=0
# 全局变量，用于存储临时变量
tmp_cv = 0

# 定义一个回调函数，用于处理滑动条的变化
def nothing(s):
    # 该函数不做任何事情，仅作为占位符
    pass

# 定义颜色范围，用于 HSV 颜色空间的阈值设置
col_black = (0,0,0,180,255,46)# black
col_red = (0,100,80,10,255,255)# red
col_blue = (100,43,46,124,255,255)# blue
col_green= (35,43,46,77,255,255)# green
col_yellow = (26,43,46,34,255,255)# yellow

# 定义颜色选择开关
Switch = '0:Red\n1:Green\n2:Blue\n3:Yellow\n4:Black'

# 定义一个名为 Follower 的类，继承自 rclpy.node.Node
# 功能：实现一个简单的线跟踪算法，通过订阅图像话题，处理图像数据，发布速度指令
class Follower(Node):
    def __init__(self):
        # 调用父类的初始化方法，将节点命名为'follower'
        super().__init__('follower')
        # 初始化 cv_bridge 对象，用于在 ROS 图像消息和 OpenCV 图像格式之间进行转换
        self.bridge = cv_bridge.CvBridge()
        # 创建服务质量配置，设置消息队列深度为 10
        qos = QoSProfile(depth=10)
        # 初始化图像矩阵
        self.mat = None
        # 创建一个订阅者，订阅'/camera/color/image_raw'主题的图像消息
```

```
        self.image_sub = self.create_subscription(
            Image,
            '/camera/color/image_raw',
            self.image_callback,
            qos)
        # 创建一个发布者，用于发布速度指令到'cmd_vel'主题
        self.cmd_vel_pub = self.create_publisher(Twist, 'cmd_vel', qos)
        # 初始化 Twist 消息
        self.twist = Twist()
        # 初始化临时变量
        self.tmp = 0

    def image_callback(self, msg):
        # 全局变量，用于存储上一次的误差值
        global last_erro
        # 全局变量，用于存储临时变量
        global tmp_cv
        # 如果是第一次接收到图像消息，创建一个窗口并添加一个滑动条
        if self.tmp==0:
            cv2.namedWindow('Adjust_hsv',cv2.WINDOW_NORMAL)
            cv2.createTrackbar(Switch,'Adjust_hsv',0,4,nothing)
            self.tmp=1
        # 将 ROS 图像消息转换为 OpenCV 图像格式
        image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
        # 将 RGB 图像转换为 HSV 颜色空间
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        # 创建一个 5x5 的卷积核
        kernel = numpy.ones((5,5),numpy.uint8)
        # 对 HSV 图像进行腐蚀操作，去除噪声
        hsv_erode = cv2.erode(hsv,kernel,iterations=1)
        # 对腐蚀后的 HSV 图像进行膨胀操作，恢复图像边缘
        hsv_dilate = cv2.dilate(hsv_erode,kernel,iterations=1)
        # 获取滑动条的当前位置
        m=cv2.getTrackbarPos(Switch,'Adjust_hsv')
        # 根据滑动条的位置选择不同的颜色范围
        if m == 0:
            lowerbH=col_red[0]
            lowerbS=col_red[1]
            lowerbV=col_red[2]
            upperbH=col_red[3]
            upperbS=col_red[4]
            upperbV=col_red[5]
        elif m == 1:
            lowerbH=col_green[0]
```

```
            lowerbS=col_green[1]
            lowerbV=col_green[2]
            upperbH=col_green[3]
            upperbS=col_green[4]
            upperbV=col_green[5]
        elif m == 2:
            lowerbH=col_blue[0]
            lowerbS=col_blue[1]
            lowerbV=col_blue[2]
            upperbH=col_blue[3]
            upperbS=col_blue[4]
            upperbV=col_blue[5]
        elif m == 3:
            lowerbH=col_yellow[0]
            lowerbS=col_yellow[1]
            lowerbV=col_yellow[2]
            upperbH=col_yellow[3]
            upperbS=col_yellow[4]
            upperbV=col_yellow[5]
        elif m == 4:
            lowerbH=col_black[0]
            lowerbS=col_black[1]
            lowerbV=col_black[2]
            upperbH=col_black[3]
            upperbS=col_black[4]
            upperbV=col_black[5]
        else:
            lowerbH=0
            lowerbS=0
            lowerbV=0
            upperbH=255
            upperbS=255
            upperbV=255

        # 根据选择的颜色范围创建一个掩码

mask=cv2.inRange(hsv_dilate,(lowerbH,lowerbS,lowerbV),(upperbH,upperbS,upperbV)
)
        # 将掩码应用到原始图像上，得到只包含目标颜色的图像
        masked = cv2.bitwise_and(image, image, mask=mask)
        # 在图像某处绘制一个指示，因为只考虑 20 行宽的图像，所以使用 numpy 切片将
以外的空间区域清空
        h, w, d = image.shape
        search_top = h-30
```

```
        search_bot = h
        mask[0:search_top, 0:w] = 0
        mask[search_bot:h, 0:w] = 0
        # 计算 mask 图像的重心，即几何中心
        M = cv2.moments(mask)
        if M['m00'] > 0:
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])
            # 计算图像中心线和目标指示线中心的距离
            erro = cx - w/2-60
            d_erro=erro-last_erro
            self.twist.linear.x = 0.11
            if erro<0:
                self.twist.angular.z = -(float(erro)*0.0011-float(d_erro)*0.0000)
#top_akm_bs
            elif erro>0:
                self.twist.angular.z = -(float(erro)*0.0011-float(d_erro)*0.0000)
#top_akm_bs
            else :
                self.twist.angular.z = 0.0
            last_erro=erro
        else:
            self.twist.linear.x = 0.0
            self.twist.angular.z = 0.0
        # 发布速度指令
        self.cmd_vel_pub.publish(self.twist)
        # 显示掩码图像
        cv2.imshow("Adjust_hsv", mask)
        # 等待 3 毫秒，用于刷新窗口显示
        cv2.waitKey(3)

# 定义主函数，程序的入口
def main(args=None):
    # 初始化 ROS 2
    rclpy.init(args=args)
    # 打印启动信息
    print('start patrolling')
    # 创建 Follower 节点实例
    follower = Follower()
    # 进入主循环，处理订阅的消息
    while rclpy.ok():
        # 一次处理一个消息
        rclpy.spin_once(follower)
        # 等待 0.1 秒，用于控制循环频率
```

```
        time.sleep(0.1)

    # 销毁节点
    follower.destroy_node()
    # 关闭 ROS 2
    rclpy.shutdown()

# 判断是否为主程序运行，如果是，则调用主函数
if __name__ == '__main__':
    main()
```

6.4 创建巡线功能包并使用

① 创建功能包

cd ~/wheeltec_ros2/src #打开工作空间下的 src 文件夹

```
wheeltec@wheeltec-virtual-machine:~$ cd wheeltec_ros2
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2$ cd src
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2/src$
```

ros2 pkg create simple_follower_ros2 --dependencies std_msgs rclcpp rclpy #在 src 文件夹下创建名为【simple_follower_ros2】的功能包，后面 std_msgs、ros2py、ros2cpp 为一些基本依赖

```
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2/src$ ros2 pkg create simple_fo
llower_ros2 --dependencies std_msgs rclcpp rclpy
/opt/ros/galactic/bin/ros2:6: DeprecationWarning: pkg_resources is deprecated as
 an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html
  from pkg_resources import load_entry_point
going to create a new package
package name: simple_follower_ros2
destination directory: /home/wheeltec/wheeltec_ros2/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['wheeltec <wheeltec@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['std_msgs', 'rclcpp', 'rclpy']
creating folder ./simple_follower_ros2
creating ./simple_follower_ros2/package.xml
creating source and include folder
creating folder ./simple_follower_ros2/src
creating folder ./simple_follower_ros2/include/simple_follower_ros2
creating ./simple_follower_ros2/CMakeLists.txt
```

cd simple_follower_ros2 #打开功能包文件夹

mkdir simple_follower_ros2 #在功能包文件夹下创建【simple_follower_ros2】文件夹，用于存放 py 文件

ls #查看文件夹下文件列表命令，以确认是否创建了【simple_follower_ros2】文件夹

```
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2/src/simple_follower_ros2$ cd ../
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2/src$ cd simple_follower_ros2/
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2/src/simple_follower_ros2$ mkdir simple_followe
r_ros2
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2/src/simple_follower_ros2$ ls
CMakeLists.txt  include  package.xml  simple_follower_ros2  src
```

把我们的例程程序【line_follow.py】复制到【simple_follower_ros2】文件夹下，或者创建文件也可以，下面演示创建文件【line_follow.py】的过程。

cd simple_follower_ros2/ #打开【simple_follower_ros2】文件夹

touch line_follow.py #创建文件【line_follow.py】

```
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2/src/simple_follower_ros2/simple_follower_ros2$
 touch line_follow.py
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2/src/simple_follower_ros2/simple_follower_ros2$
 ls
line_follow.py
```

nano line_follow.py #使用 nano 编辑器编辑文件

```
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower/scripts$ nano OpenCV
_ROS_LineFollow.py
```

复制我们的例程程序内容，在 nano 编辑器单击右键，选择粘贴。

```
        # 一次处理一个消息
        rclpy.spin_once(follower)
        # 等待0.1秒，用于控制循环频率
        time.sleep(0.1)

    # 销毁节点
    follower.destroy_node()
    # 关闭ROS 2
    rclpy.shutdown()

# 判断是否为主程序运行，如果是，则调用主函数
if __name__ == '__main__':
    main()

^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify   ^C Cur Pos
^X Exit       ^R Read File   ^\ Replace    ^U Paste Text ^T To Spell  ^_ Go To Line
```

同时按下 Ctrl+O 键，会提示【File Name to write:line_follow.py】，在按下回车键，则成功保存文件。

```
# 判断是否为主程序运行，如果是，则调用主函数
if __name__ == '__main__':
    main()

File Name to Write: line_follow.py
^C Cancel   M-M Mac Format   M-P Prepend   ^T To Files
```

再按下同时 Ctrl+X 键，退出 nano 编辑器。

sudo chmod 777 line_follow.py #给予【line_follow.py】文件可执行权限，这一步必须进行，否则会导致文件无法运行。要求输入密码为：dongguan。

② 编写 launch 文件

cd .. #返回功能包文件夹根目录

mkdir launch #创建【launch】文件夹

接下来直接复制我们的例程 launch 文件或者自行创建 launch 文件，用户可自行选择，下面简略说明创建过程。

cd launch/ #打开【launch】文件夹

touch line_follower.launch.py #创建 launch 文件

使用 nano 编辑器编辑文件，为节省篇幅不再赘述，下面是【line_follower.launch.py】文件的全部内容。

```
# 导入必要的模块
import os
import launch_ros.actions
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import (DeclareLaunchArgument, IncludeLaunchDescription)
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch.conditions import IfCondition
from launch.conditions import UnlessCondition
from launch.actions import ExecuteProcess

# 定义生成启动描述的函数
def generate_launch_description():
    # 定义一个启动参数 is_uvc_cam, 默认值为 'false'
    # 这个参数用于标识是否使用 UVC 摄像头
    is_uvc_cam = LaunchConfiguration('is_uvc_cam', default='false')

    # 获取 'turn_on_wheeltec_robot' 包的共享目录路径
    # 这里假设该包包含了机器人启动和摄像头启动的相关文件
    bringup_dir = get_package_share_directory('turn_on_wheeltec_robot')
    # 构造该包中 'launch' 文件夹的路径
    launch_dir = os.path.join(bringup_dir, 'launch')

    # 包含 wheeltec_camera.launch.py 文件的启动描述
    # 这是一个嵌套启动文件, 用于启动与摄像头相关的节点
    wheeltec_camera = IncludeLaunchDescription(
            # 指定启动文件的路径
            PythonLaunchDescriptionSource(os.path.join(launch_dir,
'wheeltec_camera.launch.py')),
    )
    # 包含 turn_on_wheeltec_robot.launch.py 文件的启动描述
```

```
    # 这是一个嵌套启动文件，用于启动机器人相关的节点
    wheeltec_robot = IncludeLaunchDescription(
            # 指定启动文件的路径
            PythonLaunchDescriptionSource(os.path.join(launch_dir,
'turn_on_wheeltec_robot.launch.py')),
    )
    # 返回一个启动描述对象，包含以下内容：
    return LaunchDescription([
        # 包含 wheeltec_robot 的启动描述
        wheeltec_robot,
        # 包含 wheeltec_camera 的启动描述
        wheeltec_camera,
        # 定义一个 ROS 2 节点
        launch_ros.actions.Node(
            # 节点所属的包名
            package='simple_follower_ros2',
            # 节点的可执行文件名
            executable='line_follow',
            # 节点的名称
            name='line_follow',
        )
    ])
```

③ 编译并运行

colcon build --packages-select simple_follower_ros2　#返回工作空间根目录后输入【colcon build --packages-select simple_follower_ros2】命令进行编译

```
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2$ colcon build --packages-sele
ct simple_follower_ros2
Starting >>> simple_follower_ros2
Finished <<< simple_follower_ros2 [9.29s]

Summary: 1 package finished [15.6s]
```

source ~/wheeltec_ros2/install/setup.bash #编译完成后输入命令 source 文件 setup.bash

```
wheeltec@wheeltec-virtual-machine:~/wheeltec_ros2$ source ~/wheeltec_ros2/insta
ll/setup.bash
```

ros2 launch simple_follower_ros2 line_follower.launch #输入命令启动 launch 文件，运行程序。

运行成功。整体效果如下

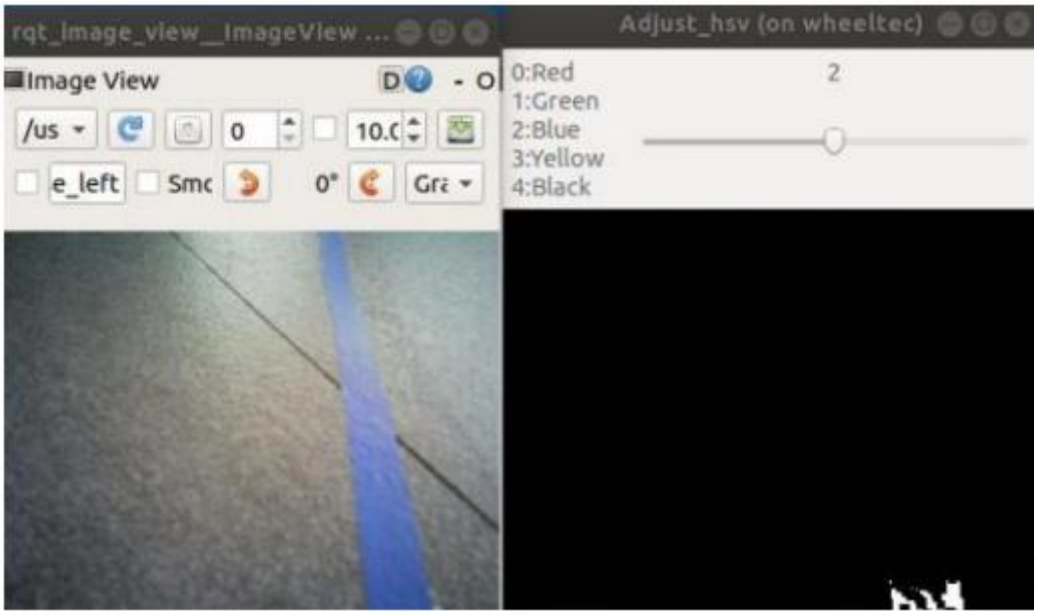

图 6-4-1 运行程序整体效果图

7. 在 ROS2 中使用 OpenCV 进行色块跟随

7.1 概述

本章的内容相对上一节巡线主要是多对深度图像的处理使用。同时本章还使用了 ROS2 参数服务器功能。

注意：本手册的例程是基于我们的 ROS2 机器人产品的，如果程序不是运行于我们的 ROS2 机器人产品上的话可能需要自行安装相关功能包。重点在于寻找轮廓与轮廓区域内的深度信息处理。

7.2 OpenCV 程序及解析

程序文件名为【visualFollower.py】和【visualTracker.py】。下面为源码及其解析。

【visualFollower.py】

```
#!/usr/bin/env python3

# 导入必要的 Python 模块
import rclpy
import _thread
import threading
import time
import numpy as np
from sensor_msgs.msg import Joy, LaserScan
from geometry_msgs.msg import Twist, Vector3
from turn_on_wheeltec_robot.msg import Position as PositionMsg
from std_msgs.msg import String as StringMsg

# 导入 ROS 2 的 Node 类和 QoSProfile
from rclpy.node import Node
from rclpy.qos import QoSProfile
from rclpy.qos import qos_profile_sensor_data

# 定义全局变量，用于存储角度和距离信息
angle=[0.0]*3
distan=[0.0]*3

# 定义 VisualFollower 类，继承自 Node
class VisualFollower(Node):
```

```
    def __init__(self):
        # 初始化父类 Node，设置节点名称为 'visualfollower'
        super().__init__('visualfollower')
        qos = QoSProfile(depth=10)  # 定义 QoSProfile，队列深度为 10

        # 定义一个定时器，用于检测控制器是否丢失连接
        # 如果 1 秒内没有收到控制器消息，则调用 controllerLoss 方法
        self.controllerLossTimer = threading.Timer(1, self.controllerLoss)
        self.controllerLossTimer.start()

        # 定义是否需要持续按住按钮的模式
        self.switchMode = True
        self.max_speed = 0.3  # 最大速度限制
        self.controllButtonIndex = -4  # 控制器按钮索引

        # 状态标志
        self.buttonCallbackBusy = False
        self.active = False
        self.i = 0

        # 创建一个发布者，用于发布速度命令到 '/cmd_vel' 主题
        self.cmdVelPublisher = self.create_publisher(Twist, '/cmd_vel', qos)

        # 创建订阅者，用于接收物体跟踪器的当前位置信息
        self.positionSubscriber = self.create_subscription(PositionMsg,
'/object_tracker/current_position', self.positionUpdateCallback, qos)
        # 创建订阅者，用于接收物体跟踪器的状态信息
        self.trackerInfoSubscriber = self.create_subscription(StringMsg,
'/object_tracker/info', self.trackerInfoCallback, qos)

        # 定义 PID 控制器的目标距离和参数
        targetDist = 600
        self.PID_controller = simplePID([0, targetDist], [1.2, 0.2], [0, 0.00],
[0.005, 0.00])

    def trackerInfoCallback(self, info):
        # 处理物体跟踪器的状态信息
        self.get_logger().warn(info.data)

    def positionUpdateCallback(self, position):
        # 处理物体跟踪器的当前位置信息
        angleX = position.angle_x  # 获取角度信息
        distance = position.distance  # 获取距离信息
```

```
        # 调用 PID 控制器更新速度
        [uncliped_ang_speed, uncliped_lin_speed] =
self.PID_controller.update([angleX, distance])
        # 限制速度范围
        angularSpeed = np.clip(-uncliped_ang_speed, -self.max_speed,
self.max_speed)
        linearSpeed = np.clip(-uncliped_lin_speed, -self.max_speed,
self.max_speed)

        # 创建 Twist 消息并发布速度命令
        velocity = Twist()
        velocity.linear.x = float(linearSpeed)
        velocity.angular.z = angularSpeed

        if (distance > 2000) or (distance == 0):
            self.stopMoving()  # 如果距离超出范围或为 0，则停止移动
        else:
            self.cmdVelPublisher.publish(velocity)

    def buttonCallback(self, joy_data):
        # 处理控制器按钮消息
        self.controllerLossTimer.cancel()  # 重置定时器
        self.controllerLossTimer = threading.Timer(0.5, self.controllerLoss)
        self.controllerLossTimer.start()

        if self.buttonCallbackBusy:
            return  # 如果正在处理按钮消息，则忽略
        else:
            _thread.start_new_thread(self.threadedButtonCallback, (joy_data,))

    def threadedButtonCallback(self, joy_data):
        # 在新线程中处理按钮消息
        self.buttonCallbackBusy = True

        if (joy_data.buttons[self.controllButtonIndex] == self.switchMode) and
self.active:
            self.get_logger().info('stopping')
            self.stopMoving()  # 停止移动
            self.active = False
        elif (joy_data.buttons[self.controllButtonIndex] == True) and not
self.active:
            self.get_logger().info('activating')
            self.active = True  # 激活跟踪
```

```
        self.buttonCallbackBusy = False

    def stopMoving(self):
        # 停止移动
        velocity = Twist()
        self.cmdVelPublisher.publish(velocity)

    def controllerLoss(self):
        # 控制器丢失连接时的处理
        self.stopMoving()
        self.active = False
        self.get_logger().info('lost connection')


class simplePID:
    '''简单的离散 PID 控制器'''
    def __init__(self, target, P, I, D):
        # 初始化 PID 控制器的参数
        self.Kp = np.array(P)
        self.Ki = np.array(I)
        self.Kd = np.array(D)
        self.setPoint = np.array(target)
        self.last_error = 0
        self.integrator = 0
        self.integrator_max = float('inf')
        self.timeOfLastCall = None

    def update(self, current_value):
        # 更新 PID 控制器
        current_value = np.array(current_value)
        error = self.setPoint - current_value

        # 对误差进行处理
        if error[0] < 0.1 and error[0] > -0.1:
            error[0] = 0
        if error[1] < 150 and error[1] > -150:
            error[1] = 0

        # 放大误差以增加速度
        if (error[1] > 0) and (self.setPoint[1] < 1200):
            error[1] = error[1] * (1200 / self.setPoint[1]) * 0.5
            error[0] = error[0] * 0.8

        P = error  # 比例项
```

```
        currentTime = time.perf_counter()
        deltaT = (currentTime - self.timeOfLastCall) if self.timeOfLastCall else
0

        # 积分项和微分项
        self.integrator += (error * deltaT)
        I = self.integrator
        D = (error - self.last_error) / deltaT if deltaT else 0

        self.last_error = error
        self.timeOfLastCall = currentTime

        # 返回控制信号
        return self.Kp * P + self.Ki * I + self.Kd * D


def main(args=None):
    # 主函数
    print('visualFollower')
    rclpy.init(args=args)  # 初始化 ROS 2
    visualFollower = VisualFollower()  # 创建 VisualFollower 节点
    print('visualFollower init done')
    try:
        rclpy.spin(visualFollower)  # 保持节点运行
    finally:
        visualFollower.destroy_node()  # 销毁节点
        rclpy.shutdown()  # 关闭 ROS 2

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

【visualTracker.py】

```
#!/usr/bin/env python3
# 使用 Python 3 解释器运行脚本

# 导入必要的模块
from __future__ import division  # 确保使用 Python 3 的除法行为
import message_filters  # 用于同步多个消息流
import numpy as np  # 用于数值计算
import cv2  # OpenCV 库，用于图像处理
import rclpy  # ROS 2 的 Python 库
from rclpy.node import Node  # ROS 2 的节点类
```

```
from matplotlib import pyplot as plt  # Matplotlib 用于绘图
import cv_bridge  # 用于在 ROS 消息和 OpenCV 图像之间进行转换

from sensor_msgs.msg import Image  # ROS 2 图像消息类型
from rclpy.qos import QoSProfile  # ROS 2 的服务质量策略
from rclpy.qos import qos_profile_sensor_data  # 预定义的传感器数据 QoS 策略

from turn_on_wheeltec_robot.msg import Position as PositionMsg  # 自定义消息类型
from std_msgs.msg import String as StringMsg  # 标准字符串消息类型

# 设置 NumPy 的错误处理方式，确保所有错误都会抛出异常
np.seterr(all='raise')
displayImage = False  # 是否显示调试图像
plt.close('all')  # 关闭所有 Matplotlib 图形窗口

# 定义 VisualTracker 类，继承自 Node
class VisualTracker(Node):
    def __init__(self):
        # 初始化父类 Node，设置节点名称为 'visualtracker'
        super().__init__('visualtracker')
        qos = QoSProfile(depth=10)  # 定义 QoS 策略，队列深度为 10
        self.bridge = cv_bridge.CvBridge()  # 创建 CvBridge 对象，用于消息和图像
之间的转换

        # 定义目标颜色的 HSV 范围（红色）
        self.targetUpper = np.array([0, 50, 50])
        self.targetLower = np.array([180, 255, 255])

        # 定义图像的尺寸和相机的视角角度
        self.pictureHeight = 480
        self.pictureWidth = 640
        vertAngle = 0.43196898986859655  # 垂直视角角度（弧度）
        horizontalAngle = 0.5235987755982988  # 水平视角角度（弧度）

        # 预计算正切值，用于后续角度计算
        self.tanVertical = np.tan(vertAngle)
        self.tanHorizontal = np.tan(horizontalAngle)
        self.lastPosition = None  # 上一次检测到的目标位置

        # 创建消息过滤器，分别订阅 RGB 图像和深度图像
        im_sub = message_filters.Subscriber(self, Image,
'/camera/color/image_raw')
        dep_sub = message_filters.Subscriber(self, Image,
'/camera/depth/image_raw', qos_profile=qos_profile_sensor_data)
```

```
        queue_size = 30  # 消息队列大小

        # 使用时间同步器同步 RGB 和深度图像
        self.timeSynchronizer =
message_filters.ApproximateTimeSynchronizer([im_sub, dep_sub], queue_size, 0.05)
        self.timeSynchronizer.registerCallback(self.trackObject)  # 注册回调函
数

        # 创建发布者，用于发布目标位置信息
        self.positionPublisher = self.create_publisher(PositionMsg,
'/object_tracker/current_position', qos)
        self.posMsg = PositionMsg()  # 创建自定义消息对象

    def trackObject(self, image_data, depth_data):
        # 回调函数：处理 RGB 图像和深度图像
        if image_data.encoding != 'rgb8':
            raise ValueError('image is not rgb8 as expected')  # 检查图像编码是
否为 rgb8

        # 将 ROS 图像消息转换为 OpenCV 图像
        frame = self.bridge.imgmsg_to_cv2(image_data, desired_encoding='rgb8')
        depthFrame = self.bridge.imgmsg_to_cv2(depth_data,
desired_encoding='passthrough')

        # 检查图像尺寸是否符合预期
        if np.shape(frame)[0:2] != (self.pictureHeight, self.pictureWidth):
            raise ValueError('image does not have the right shape')

        # 将 RGB 图像转换为 HSV 色彩空间
        hsv = cv2.cvtColor(frame, cv2.COLOR_RGB2HSV)

        # 根据目标颜色范围创建掩码
        org_mask = cv2.inRange(hsv, self.targetLower, self.targetUpper)

        # 对掩码进行形态学操作（腐蚀和膨胀），去除噪声
        mask = cv2.erode(org_mask, None, iterations=4)

        # 查找目标轮廓
        contours = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]

        newPos = None  # 初始化目标位置变量
        if displayImage:  # 如果启用调试图像显示
            # 显示原始图像、掩码图像等
```

```
            backConverted = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
            plt.figure()
            plt.subplot(2, 2, 1)
            plt.imshow(frame)
            plt.xticks([]), plt.yticks([])
            plt.subplot(2, 2, 2)
            plt.imshow(org_mask, cmap='gray', interpolation='bicubic')
            plt.xticks([]), plt.yticks([])
            plt.subplot(2, 2, 3)
            plt.imshow(mask, cmap='gray', interpolation='bicubic')
            plt.xticks([]), plt.yticks([])
            plt.show()
            rclpy.sleep(0.2)

        # 遍历轮廓，寻找目标
        for contour in sorted(contours, key=cv2.contourArea, reverse=True):
            pos = self.analyseContour(contour, depthFrame)  # 分析轮廓，获取目标
位置
            if newPos is None:
                newPos = pos  # 如果未找到目标，则使用当前轮廓作为候选
            self.lastPosition = pos  # 更新上一次检测到的目标位置
            self.publishPosition(pos)  # 发布目标位置
            return

        self.lastPosition = newPos  # 如果未找到目标，则更新目标位置为 None

    def publishPosition(self, pos):
        # 发布目标位置信息
        self.posMsg.angle_x = self.calculateAngleX(pos)  # 计算水平角度
        self.posMsg.angle_y = self.calculateAngleY(pos)  # 计算垂直角度
        self.posMsg.distance = float(pos[1])  # 设置目标距离
        self.positionPublisher.publish(self.posMsg)  # 发布消息

    def calculateAngleX(self, pos):
        # 计算目标的水平角度
        centerX = pos[0][0]
        displacement = 2 * centerX / self.pictureWidth - 1
        angle = -1 * np.arctan(displacement * self.tanHorizontal)
        return angle

    def calculateAngleY(self, pos):
        # 计算目标的垂直角度
        centerY = pos[0][1]
        displacement = 2 * centerY / self.pictureHeight - 1
```

```
        angle = -1 * np.arctan(displacement * self.tanVertical)
        return angle

    def analyseContour(self, contour, depthFrame):
        # 分析轮廓，获取目标的中心坐标和距离
        centerRaw, size, rotation = cv2.minAreaRect(contour)  # 获取最小外接矩形
        center = np.round(centerRaw).astype(int)  # 获取矩形中心

        # 获取目标区域的深度值
        minSize = int(min(size) / 3)
        depthObject = depthFrame[(center[1] - minSize):(center[1] + minSize),
(center[0] - minSize):(center[0] + minSize)]
        depthArray = depthObject[~np.isnan(depthObject)]

        if len(depthArray) == 0:
            self.get_logger().warn('empty depth array. all depth values are nan')
# 如果深度值为空，记录警告
            averageDistance = 0
        else:
            averageDistance = np.mean(depthArray)  # 计算目标的平均距离

        return (centerRaw, averageDistance)  # 返回目标的中心坐标和距离

def main(args=None):
    # 主函数
    print('visual_tracker')
    rclpy.init(args=args)  # 初始化 ROS 2
    visualTracker = VisualTracker()  # 创建 VisualTracker 节点
    print('visualTracker init done')
    try:
        rclpy.spin(visualTracker)  # 保持节点运行
    finally:
        visualTracker.destroy_node()  # 销毁节点
        rclpy.shutdown()  # 关闭 ROS 2

if __name__ == '__main__':
    main()  # 执行主函数
```

7.3 launch 文件(使用参数服务器)

文件名为【visual_follower.launch】，使用方法与巡线例程一样，可以直接存放在巡线例程的功能包内，这里不再赘述。

① visual_follower.launch

```
# 导入必要的模块
import os
from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.actions import (DeclareLaunchArgument, GroupAction,
                            IncludeLaunchDescription, SetEnvironmentVariable)
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
import launch_ros.actions
from launch.conditions import IfCondition

# 定义生成启动描述的函数
def generate_launch_description():
    # 获取 'turn_on_wheeltec_robot' 包的共享目录路径
    bringup_dir = get_package_share_directory('turn_on_wheeltec_robot')
    # 构造该包中 'launch' 文件夹的路径
    launch_dir = os.path.join(bringup_dir, 'launch')

    # 包含 'turn_on_wheeltec_robot.launch.py' 文件的启动描述
    # 用于启动 Wheeltec 机器人的相关节点
    wheeltec_robot = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(launch_dir,
'turn_on_wheeltec_robot.launch.py')),
    )

    # 包含 'wheeltec_camera.launch.py' 文件的启动描述
    # 用于启动 Wheeltec 摄像头的相关节点
    wheeltec_camera = IncludeLaunchDescription(
            PythonLaunchDescriptionSource(os.path.join(launch_dir,
'wheeltec_camera.launch.py')),
    )

    # 返回一个启动描述对象，包含以下内容：
    return LaunchDescription([
        # 启动 Wheeltec 机器人相关的节点
        wheeltec_robot,
        # 启动 Wheeltec 摄像头相关的节点
        wheeltec_camera,

        # 启动一个名为 'visualtracker' 的节点
        # 该节点属于 'simple_follower_ros2' 包，可执行文件名为 'visualtracker'
        launch_ros.actions.Node(
```

```
            package='simple_follower_ros2',
            executable='visualtracker',
            name='visualtracker',
        ),

        # 启动一个名为 'visualfollow' 的节点
        # 该节点属于 'simple_follower_ros2' 包，可执行文件名为 'visualfollow'
        # 输出设置为 'screen'，表示将日志输出到终端
        launch_ros.actions.Node(
            package='simple_follower_ros2',
            executable='visualfollow',
            name='visualfollow',
            output='screen',
        ),
    ])
```

② 启动效果

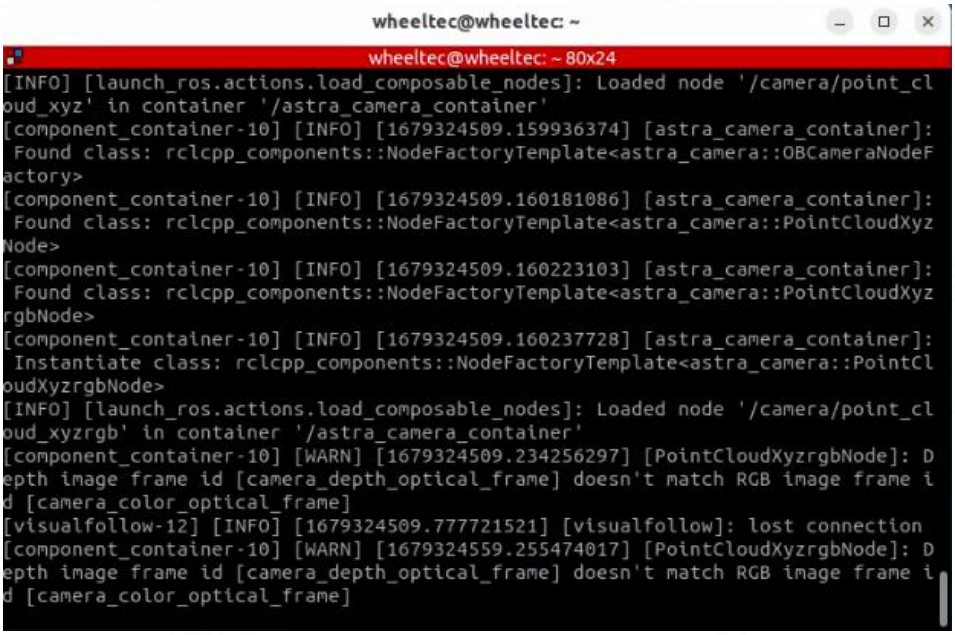

图 7-3-1 运行效果

7.4 注意事项

由于视觉跟随功能启动后，会不断寻找最近红色物体，如想对单一物体进行跟随，请尽量保持周围环境杂物不要过多以免物体被遮挡无法识别到颜色，红色色系杂物尽量不放在视野内

如果使用的是 Gemini 和 Dabai 相机，可以在视觉跟随启动文件中修改 Height 为 400，如图所示

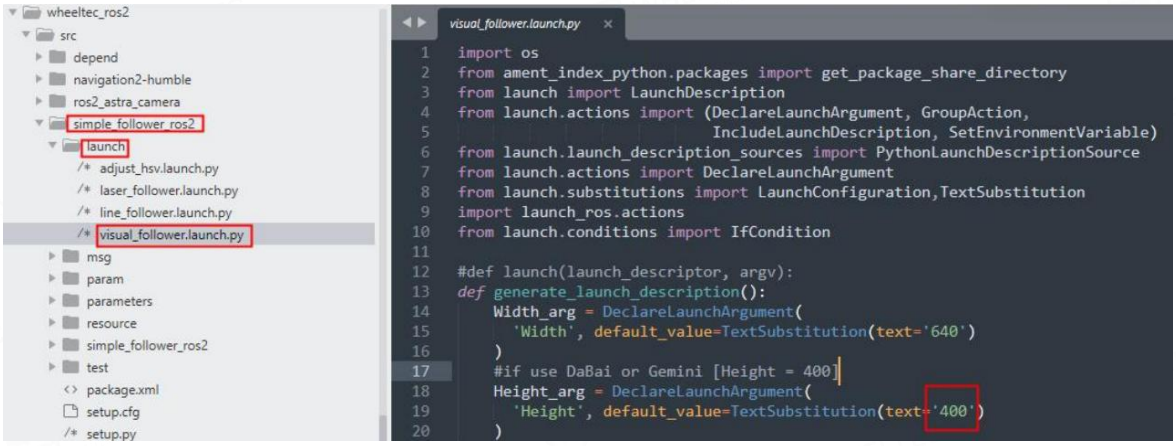

图 7-4-1 修改路径