轮 趣 科 技

OpenCV 入门及其在 ROS 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	2020/10/29	第一次发布
V3.5	2021/03/03	更正关于编译器声明的说明

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 ROS 小车产品，例如麦克纳姆轮 ROS 小车、阿克曼 ROS 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 ROS 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 ROS 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 ROS 环境下的 opencv。

sudo apt-get install python-opencv

sudo apt-get install ros-melodic-vision-opencv libopencv-dev python-opencv

注意：本节除 ROS 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。

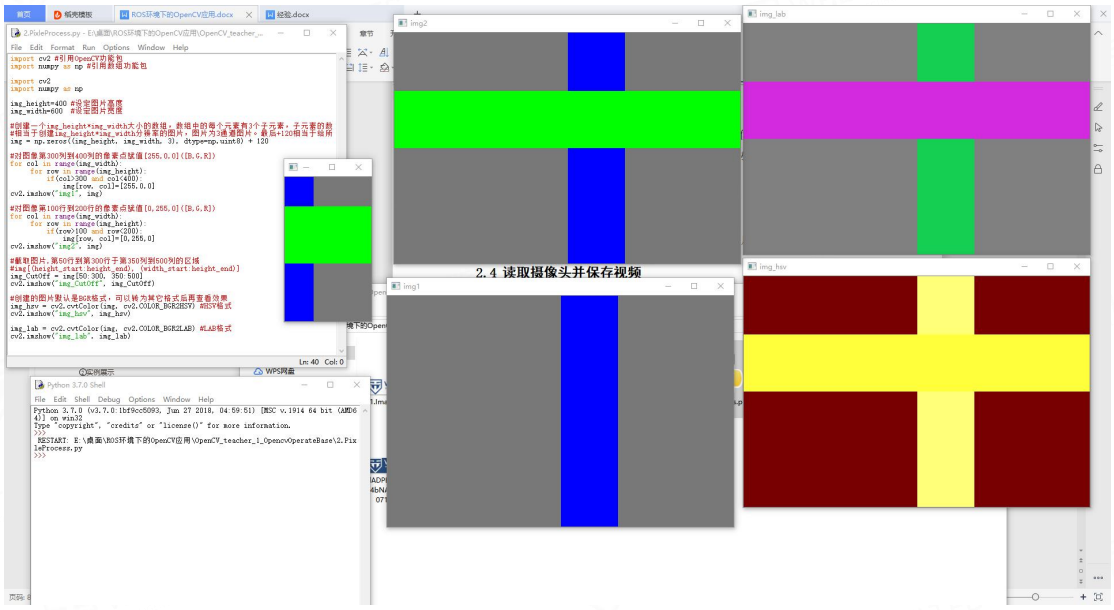


图 2-2-1【2.PixleProcess.py】运行效果

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.zeros((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。

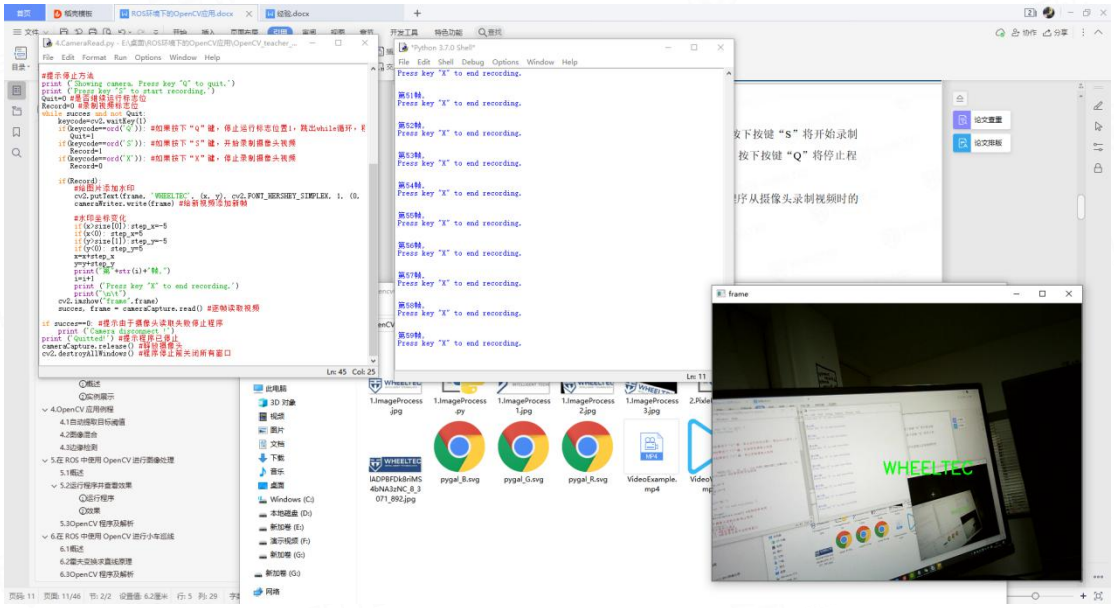


图 2-4-1 【4.CameraRead.py】运行效果

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

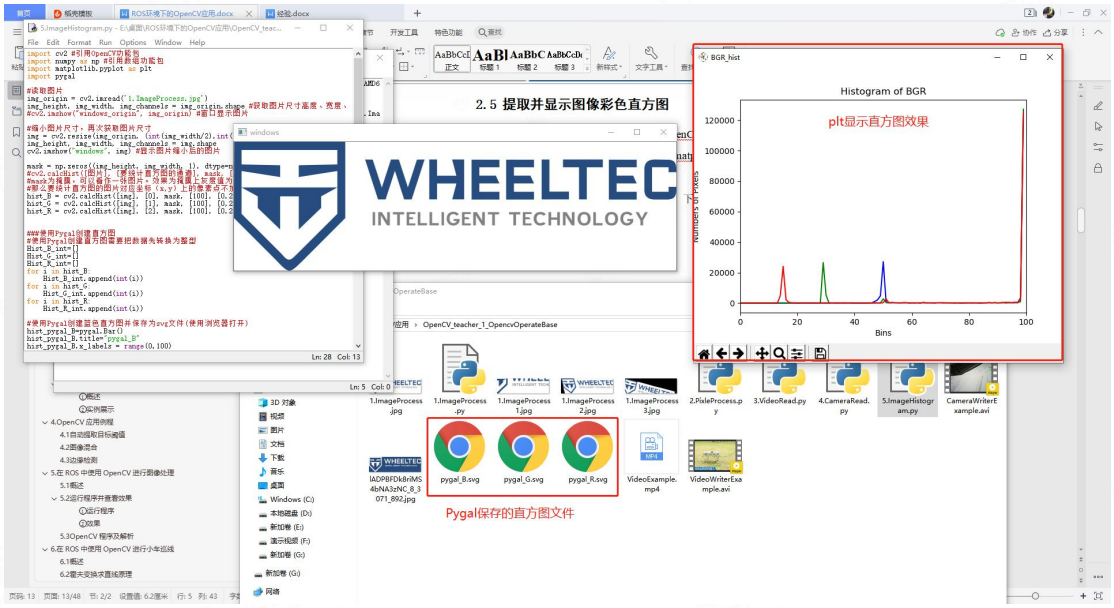


图 2-5-1 【5.ImageHistogram.py】运行效果

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.zeros((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。

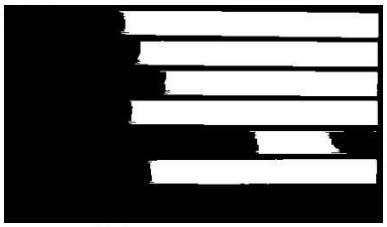


图 3-1-1 黑白图像

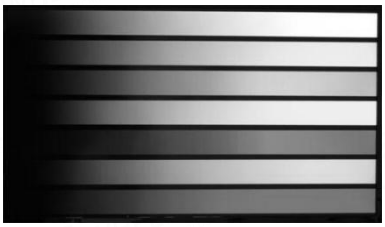


图 3-1-2 灰度图像

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。

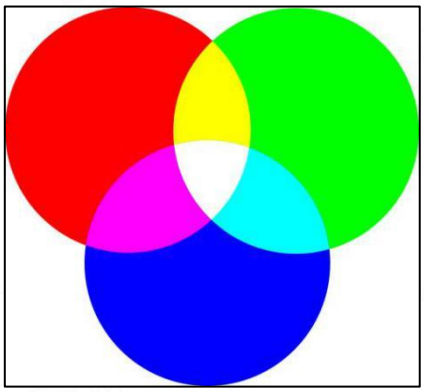


图 3-1-3 RGB 彩色图像

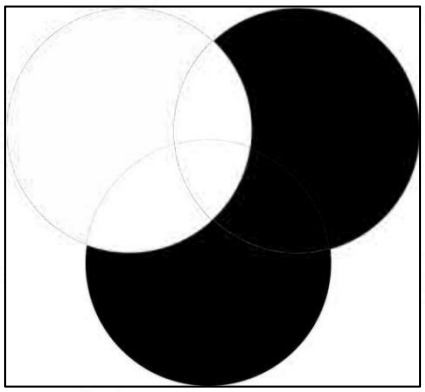


图 3-1-4 RGB 彩色图像的 R 通道

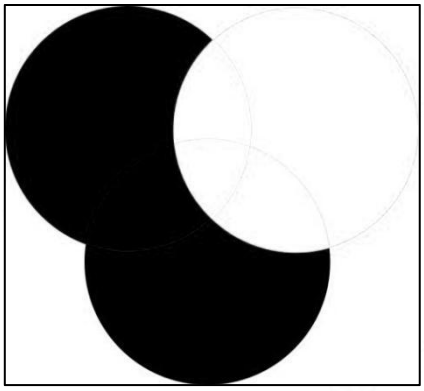


图 3-1-5 RGB 彩色图像的 G 通道

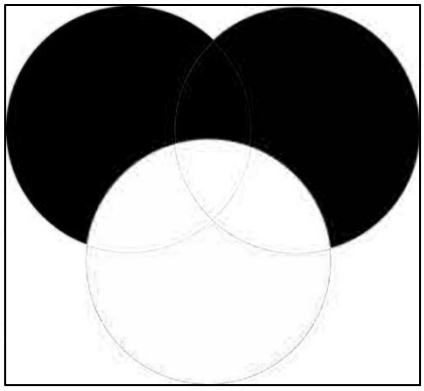


图 3-1-6 RGB 彩色图像的 G 通道

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。

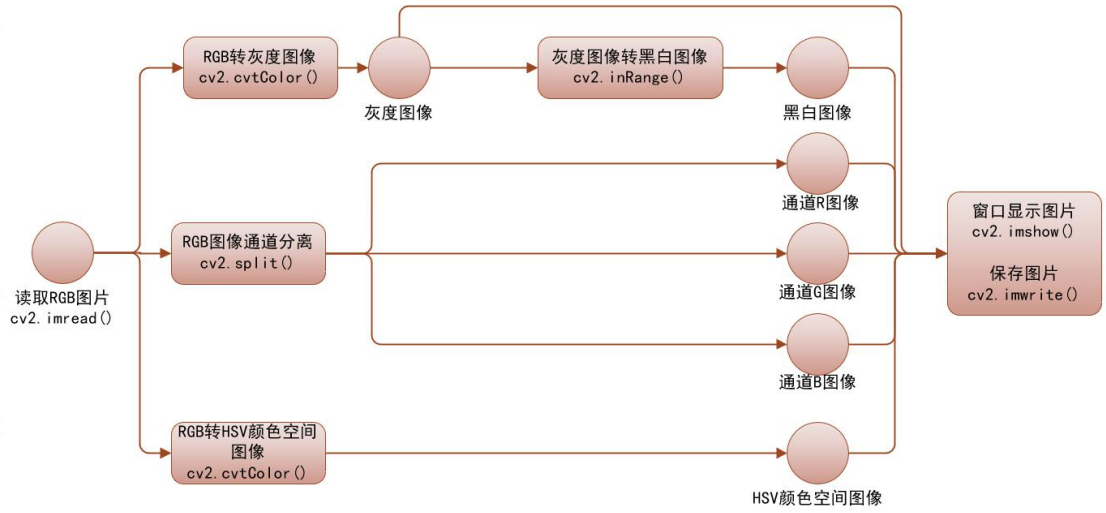


图 3-1-7 程序例程执行流程图

```
#!/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。

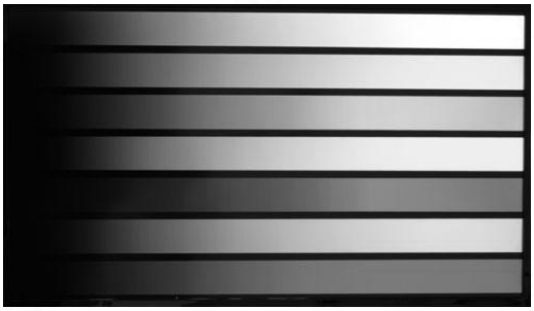


图 3-2-1 灰度图像

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。

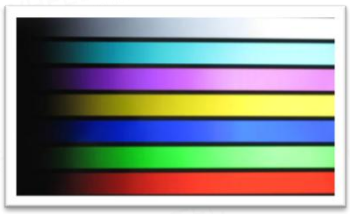


图 3-2-5 彩色图像

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。

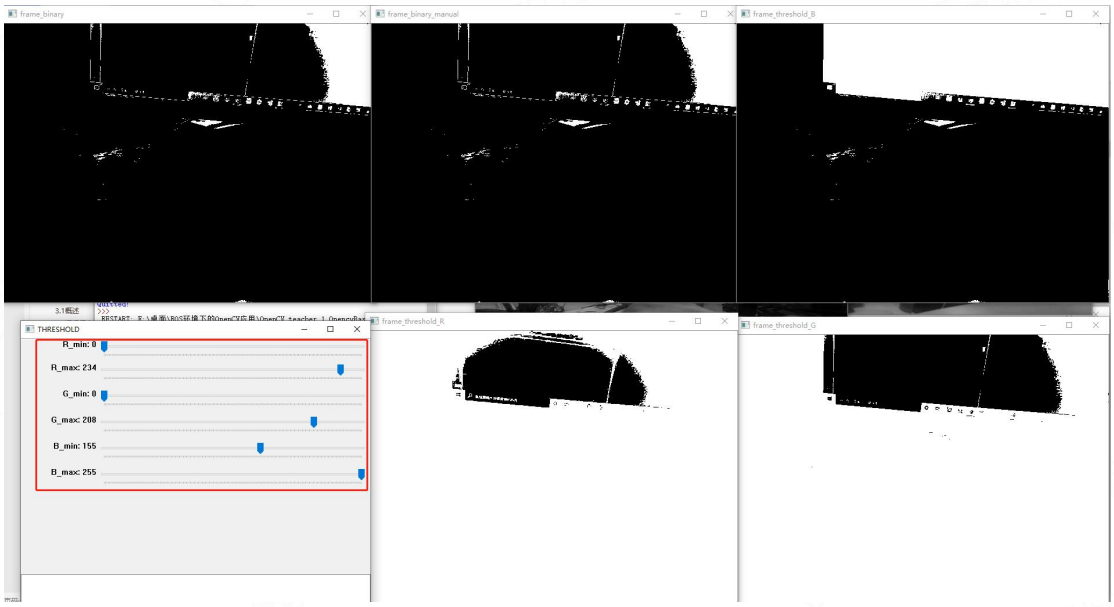


图 3-2-10 例程程序运行效果图

```
#!/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.zeros((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。

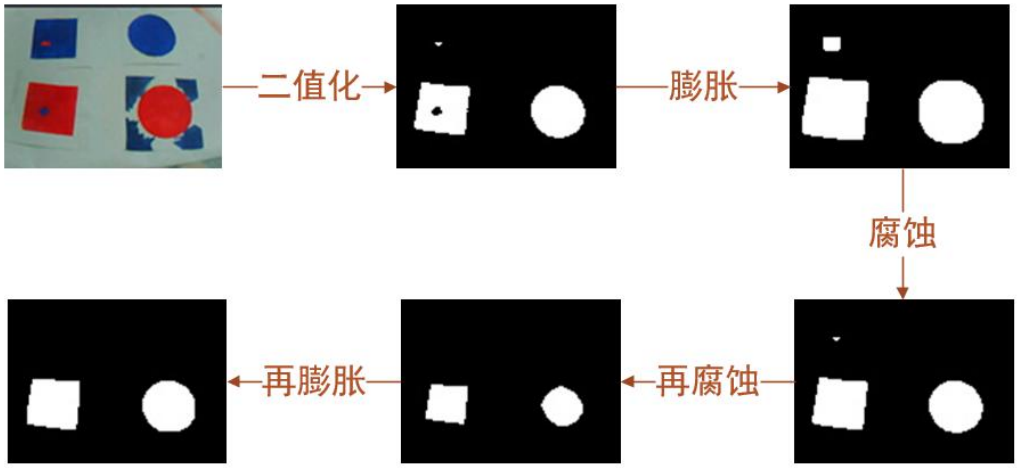


图 3-3-1 膨胀腐蚀实例

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。

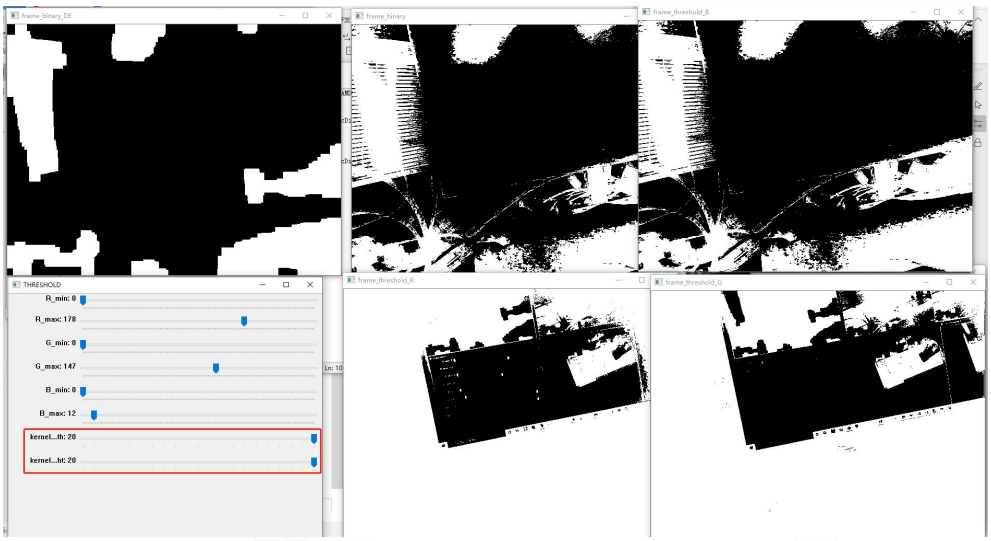


图 3-2-2 例程程序运行效果图

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】。

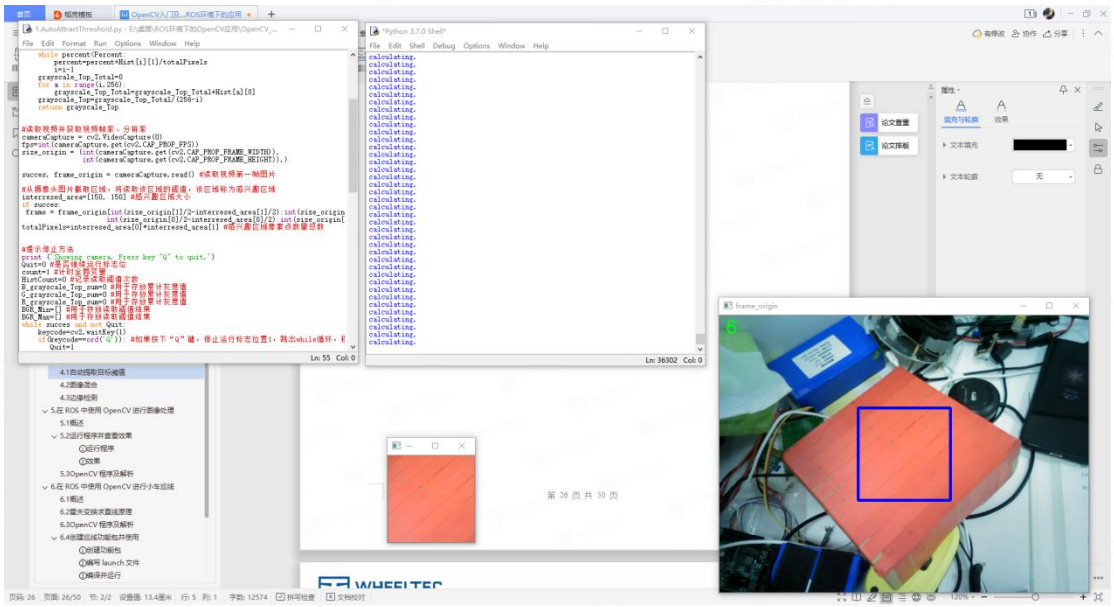


图 4-1-1 程序正在提取阈值

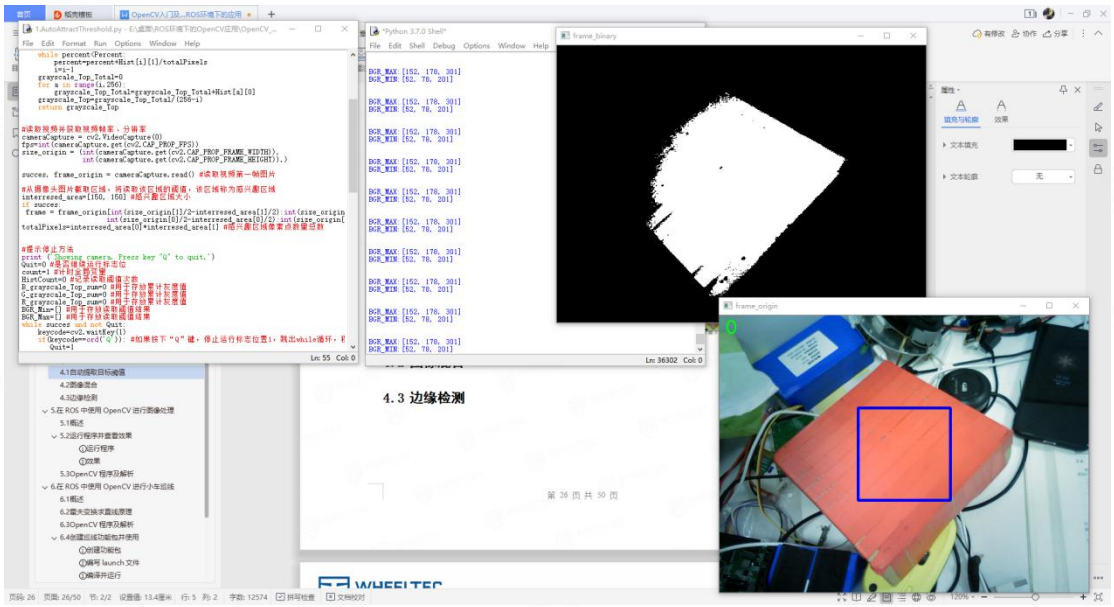


图 4-1-2 自动提取阈值的二值化效果

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.zeros((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.zeros((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.zeros((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。

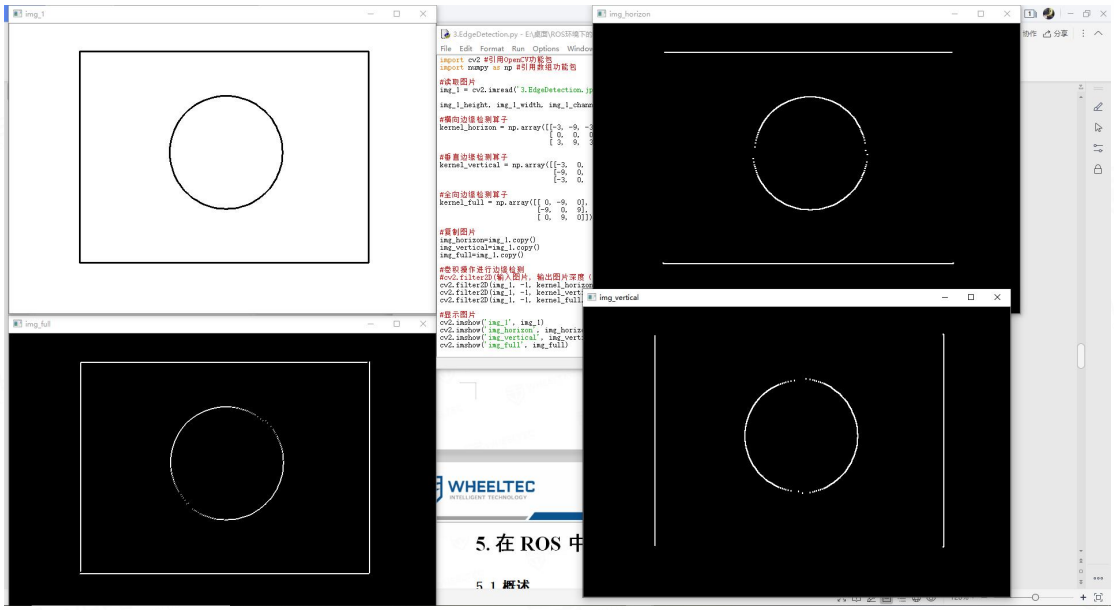


图 4-5-3 程序运行效果图

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 ROS 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 ROS 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。

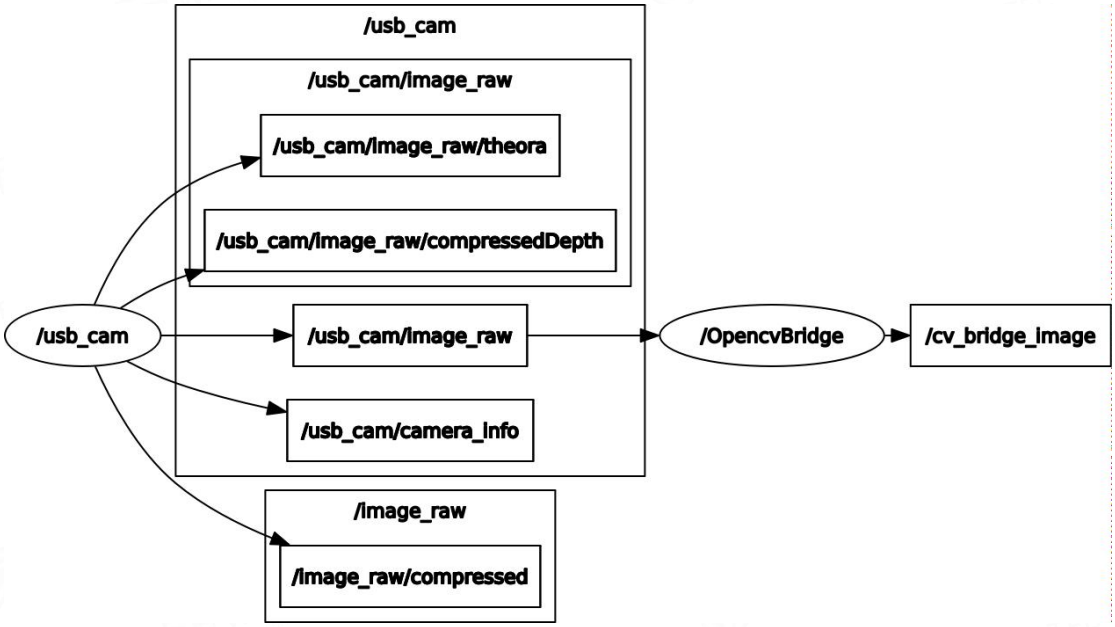


图 5-1-1 例程节点与话题通信架构

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【/usb_cam】发布了很多话题，其中包括【/usb_cam/image_raw】，节点【/usb_cam】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【/OpencvBridge】订阅了话题【/usb_cam/image_raw】，以及发布了话题【/cv_bridge_image】，本章的重点就是这个【/OpencvBridge】节点。

5.2 运行程序并查看效果

① 运行程序

roslaunch usb_cam usb_cam-test.launch #启动【/usb_cam】节点

python open_cv_test.py #运行程序启动【/OpencvBridge】节点

② 效果

运行 open_cv_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。

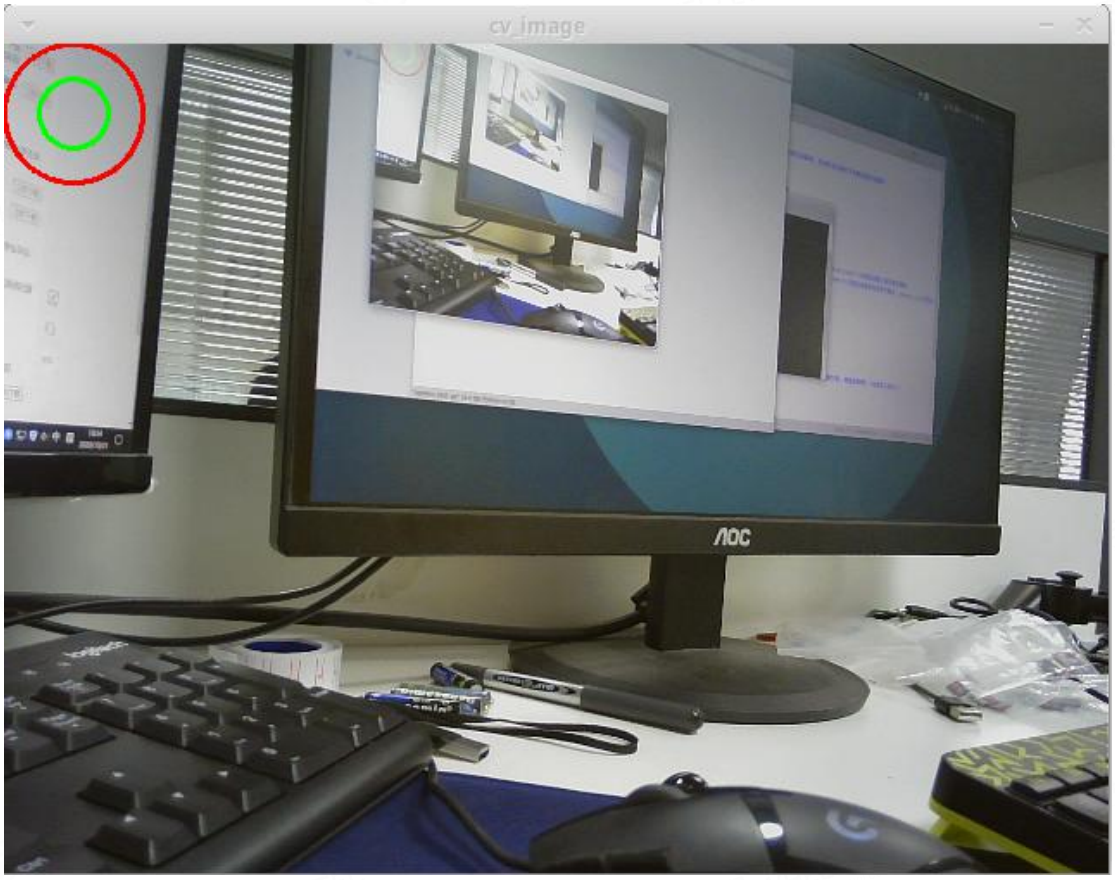


图 5-2-1

打开新终端输入：rqt_image_view。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【/usb_cam】发布的话题【/usb_cam/image_raw】和节点【/OpencvBridge】发布的话题【/cv_bridge_image】。

其中【/usb_cam/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。

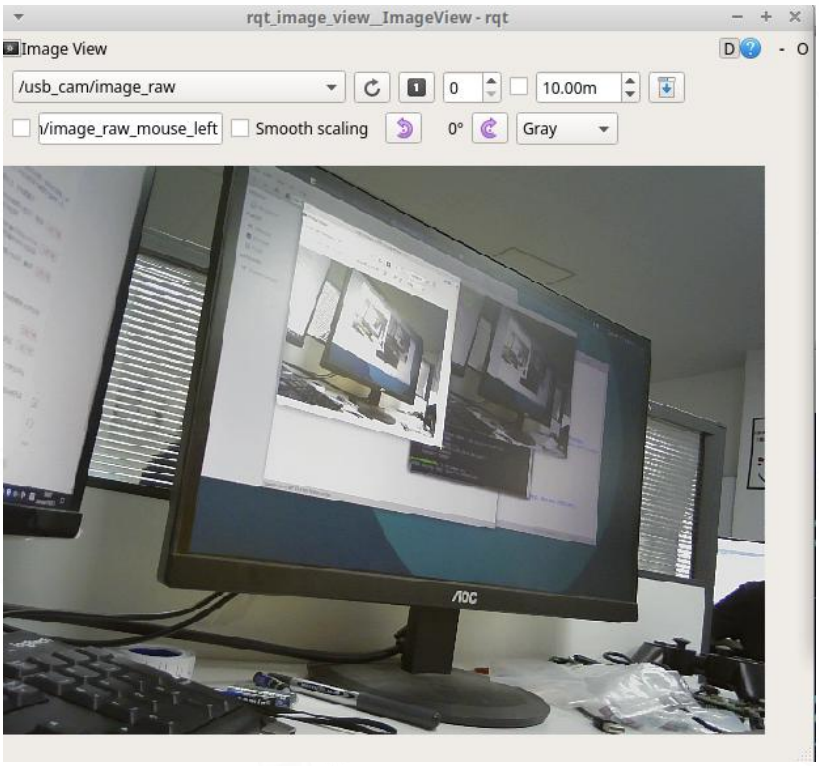


图 5-2-2 【/usb_cam/image_raw】

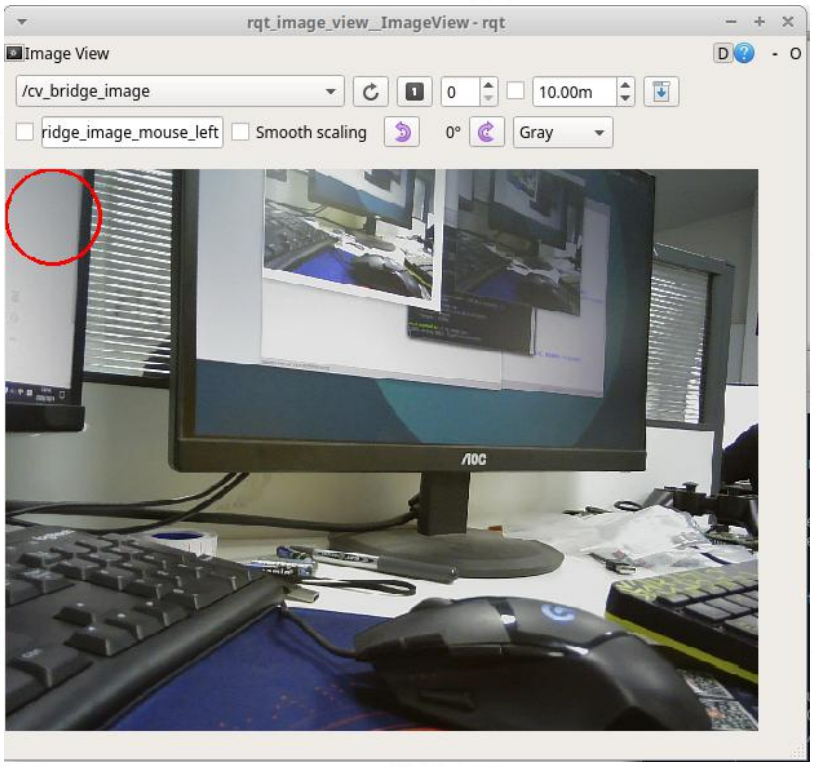


图 5-2-3 【/cv_bridge_image】

5.3 OpenCV 程序及解析

【/usb_cam】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【open_cv_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。

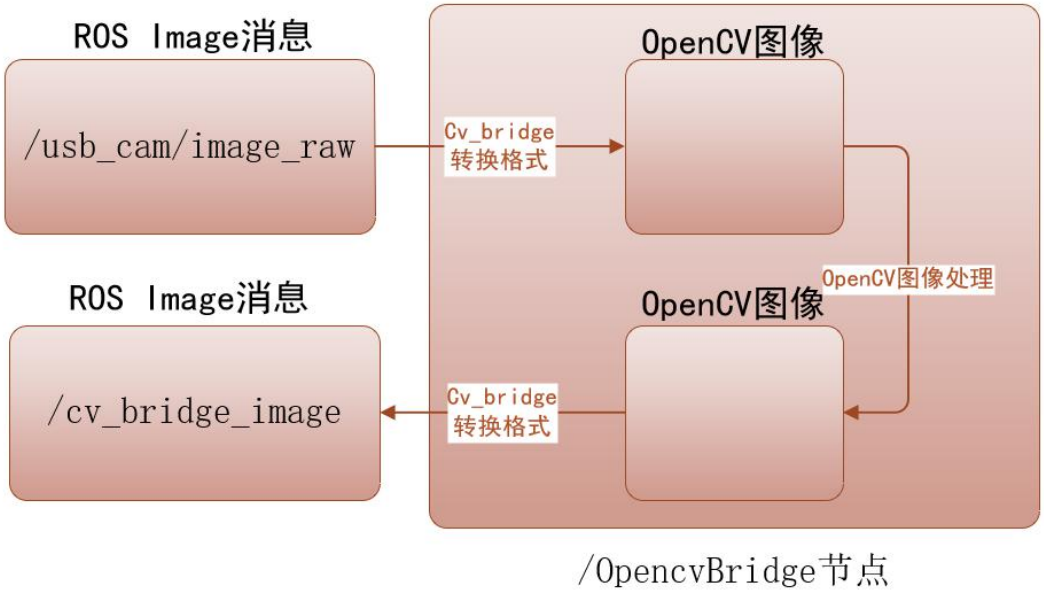


图 5-3-1 程序执行流程图

以下为【open_cv_test.py】的源码与解析。

```
#!/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 rospy #引用 ROS 的 Python 接口功能包
import cv2, cv_bridge #引用 opencv 功能包。cv_bridge 是 ROS 图像消息和 OpenCV 图像之间转换的功能包
from sensor_msgs.msg import Image #引用 ROS 内的图片消息格式

#定义一个图片转换的类，功能为：订阅 ROS 图片消息并转换为 OpenCV 格式处理，处理完成再转换回 ROS 图片消息后发布
class Image_converter:
 def __init__(self): #类成员初始化函数
     self.bridge = cv_bridge.CvBridge() #初始化图片转换功能，cv_bridge.CvBridge()
     self.image_sub = rospy.Subscriber("/usb_cam/image_raw", Image, self.callback) #初始化订阅者,rospy.Publisher()功能是创建订阅者类并输出
     self.image_pub = rospy.Publisher("cv_bridge_image", Image, queue_size=1) #初始化发布者,rospy.Publisher()功能是创建发布者类并输出。queue_size 为队列长度,
```

```
当消息发布后订阅者暂时没有接收处理，则该消息进入缓存循环发送，当队列满后最老的数据被踢出队列

 def callback(self, data): #订阅者接受到消息后的回调函数
     try: #尝试把订阅到的消息转换为 opencv 图片格式
      cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
     except CvBridgeError as e: #转换失败则把错误打印出来
      print e

     (rows, cols, channels) = cv_image.shape #获得转换后图片的宽度、高度和图片通道数
     if cols > 40 and rows > 40 :#如果图片宽高都大于 60，则在图片坐标(60, 60)的位置画一个圆。图片左上角为坐标原点
      cv2.circle(cv_image, (40, 40), 40, (0, 0, 255), 2) #API 入口参数说明:(图片, XY 坐标, 圆半径大小 ,圆 BGR 颜色值, 圆线条粗细(-1 代表实心填充))

     try: #尝试把 opencv 图片转换为 ROS 图片消息格式并发布
      self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_image, "bgr8"))
     except CvBridgeError as e: #转换失败则把错误打印出来
      print e

     #直接使用 OpenCV 创建窗口显示图片
     cv2.circle(cv_image, (40, 40), 20, (0, 255, 0), 2) #再画一个圆
     cv2.imshow("cv_image", cv_image) #OpenCV 创建窗口显示图片
     cv2.waitKey(10) #功能为刷新图像，若要创建窗口显示图片必须有这一函数。入口参数为延时时间，单位 ms，为 0 则无限延时。函数返回延时时间内键盘按键的 ASCII 码值

if __name__ == '__main__': #这段判断的作用是，如果本 py 文件是直接运行的则判断通过执行 if 内的内容，如果是 import 到其他的 py 文件中被调用(模块重用)则判断不通过
  rospy.init_node("OpencvBridge") #创建节点
  rospy.loginfo("cv_bridge_test node started") #打印 ROS 消息说明节点已开始运行
  Image_converter() #直接运行 image_converter()函数创建类，该类在运行期间会一直存在。因为该类没有需要调用的函数，所以使用赋值的形式: a=image_converter()
  rospy.spin() #相当于 while(1),当订阅者接收到新消息时调用回调函数
```

6. 在 ROS 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：霍夫变换，本章巡线使用到的直线识别 API 就是基于霍夫变换原理的。下图 4-1-1 为本章巡线的大致流程。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 ROS 机器人产品的，如果程序不是运行于我们的 ROS 机器人产品上的话可能需要自行安装相关功能包。

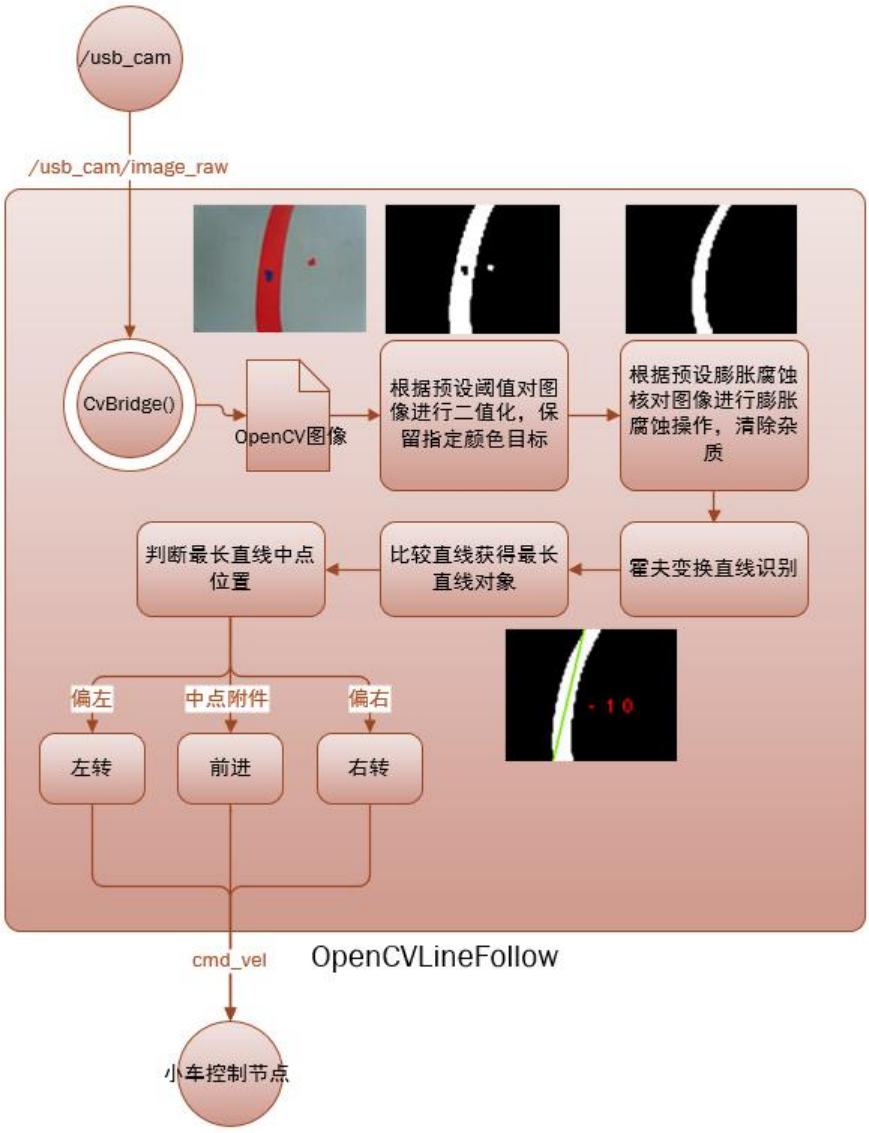


图 6-1-1 ROS 下的 OpenCV 巡线过程

6.2 霍夫变换求直线原理

霍夫变换作为一种特征检测技术，在图像分析、计算机视觉领域中有广泛应用，用于识别寻找物体中的特征，例如：线条。它的识别过程大致如下，确定要识别的形状的类型，算法会在参数空间中执行投票来决定物体的形状，在参数空间中哪个点或者区域被投票最多，则这个点或者区域就代表了形状的属性。

设在直角坐标系 x-y 有一条直线：

l_a：$y = k \times x + b$ ，取 $k = 2$、$b = 5$ 时它在 x-y 坐标系如右图所示。

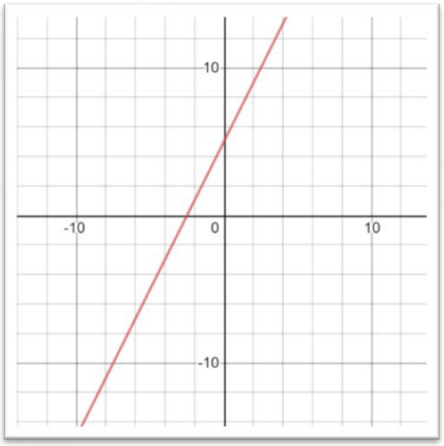


图 6-2-1 y=2*x+5

我们把直线l_a公式转换一下得到 $b = -x \times k + y$。即在直角坐标系 b-k 中，以 x、y 为参数画直线，最后我们发现，在 b-k 直角坐标系中这些直线都相交与一点(-2, 5)=(-k, b)。

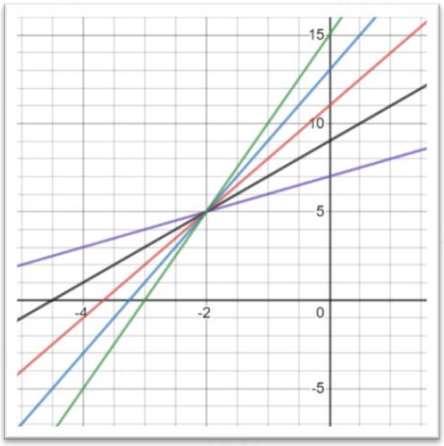


图 6-2-2 b=-x*k+y，交点(-2, 5)

即直线l_a在参数空间 b-k 中，在点(-2， 5)被投票最多，该点则代表了直线l_a的属性。

当在 x-y 坐标系中直线垂直于 x 轴即 k=∞时，在 k-b 参数空间是不方便表示的，因此一般参数空间不采用直角坐标系而是采用极坐标系(ρ-θ)：$\rho = x * \times \cos\theta + y \times \sin\theta$ 。(OpenCV 的霍夫直线检测的参数空间也是采用的极坐标系)。

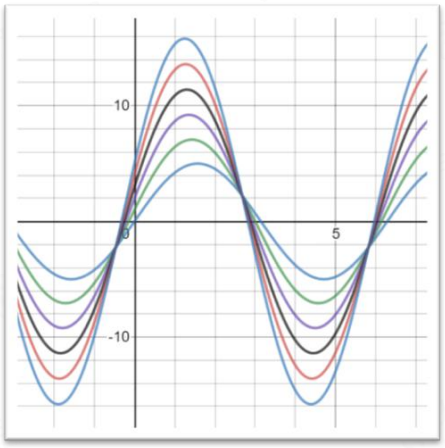


图 6-2-3 $\rho = x * \times \cos\theta + y \times \sin\theta$

设直线l_a在极坐标参数空间中的两条曲线分别为：

$$\rho = x_1 * \times \cos\theta + y_1 \times \sin\theta$$

$$\rho = x_2 * \times \cos\theta + y_2 \times \sin\theta$$

经过计算，可以求出交点坐标为$(\theta, \rho) = (\theta_a - 90^{\circ}, -b \times \cos\theta_a)$，其中$\theta_a = \arctan k$。

将 k=2，b=5，代入上式得$(\theta, \rho) = (-0.464, -2.236)$。

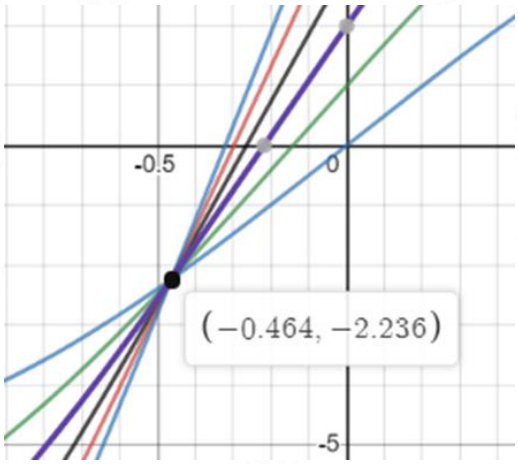


图 6-2-4 交点(-0.464，-2.236)

6.3 OpenCV 程序及解析

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

```
#!/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 rospy #引用ROS的Python接口功能包
import cv2, cv_bridge #引用opencv功能包。cv_bridge是ROS图像消息和OpenCV图像之间转换的功能包
import numpy as np #引用数组功能包
from sensor_msgs.msg import Image #引用ROS内的图片消息格式
from geometry_msgs.msg import Twist #引用ROS的速度消息格式

'''
#调阈值回调函数
def Thresholdcallback(object):
    pass

#创建画布、窗口、进度条
canvas = np.zeros((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,Thresholdcallback) #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,Thresholdcallback)
cv2.createTrackbar("G_min","THRESHOLD",0,255,Thresholdcallback)
cv2.createTrackbar("G_max","THRESHOLD",0,255,Thresholdcallback)
cv2.createTrackbar("B_min","THRESHOLD",0,255,Thresholdcallback)
cv2.createTrackbar("B_max","THRESHOLD",0,255,Thresholdcallback)
cv2.createTrackbar("kernel_width","THRESHOLD",1,20,Thresholdcallback)
cv2.createTrackbar("kernel_height","THRESHOLD",1,20,Thresholdcallback)'''
#动态调节窗口在调试完成得到合适的阈值参数后应该注释掉，以节省系统资源

#定义一个图片转换的类，功能为：订阅ROS图片消息并转换为OpenCV格式处理，处理完成再转换回ROS图片消息后发布
class Line_Follow:
 def __init__(self): #类成员初始化函数
     self.bridge = cv_bridge.CvBridge() #初始化图片转换功能，cv_bridge.CvBridge()
     self.image_sub = rospy.Subscriber("/usb_cam/image_raw", Image, self.callback)
#初始化订阅者,rospy.Publisher()功能是创建订阅者类并输出
     self.cmd_vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=1) #初始化发布者,发表话题名为“cmd_vel”,该名与其它节点订阅的控制命令话题要求一致，消息格
```

```
式为 Twist 也其它节点订阅的控制命令话题消息格式要求一致
    self.twist = Twist() #创建速度控制命令变量

def callback(self, data): #订阅者接受到消息后的回调函数
    cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8") #ROS 图片消息转换为 OpenCV
图片格式
    cv_image = cv2.resize(cv_image, (320,240), interpolation=cv2.INTER_AREA) #
压缩图片，降低图片分辨率为 320*240。这一段很重要，分辨率太高会导致图像处理需要的
算力大增，降低实时性，从而降低巡线效果
    (rows, cols, channels) = cv_image.shape #获得转换后图片的宽度、高度和图片通
道数

    #进度条值赋值相关变量
    '''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",)
    kernel_width=cv2.getTrackbarPos("kernel_width","THRESHOLD",)
    kernel_height=cv2.getTrackbarPos("kernel_height","THRESHOLD",)
    if(kernel_width<1):kernel_width=1
    if(kernel_height<1):kernel_height=1'''
    #动态调节窗口在调试完成得到合适的阈值参数后应该注释掉，以节省系统资源

    #使用数组进行彩色图像二值化
    #lower_rgb = np.array([B_min, G_min, R_min])
    #upper_rgb = np.array([B_max, G_max, R_max])
    lower_rgb = np.array([140, 40, 20])   #调试得到的合适的阈值，蓝色
    upper_rgb = np.array([230, 160, 130]) #调试得到的合适的阈值，蓝色
    image_threshold = cv2.inRange(cv_image,lower_rgb,upper_rgb)

    kernel_width=5;  #调试得到的合适的膨胀腐蚀核大小
    kernel_height=5; #调试得到的合适的膨胀腐蚀核大小
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_width,
kernel_height))
    image_threshold = cv2.erode(image_threshold, kernel)
    image_threshold = cv2.dilate(image_threshold, kernel)
    image_threshold = cv2.dilate(image_threshold, kernel)
    image_threshold = cv2.erode(image_threshold, kernel)

    #使用 OpenCV 的 API 进行霍夫变换直线识别
    Rho=1 #生成极坐标时的像素扫描步长
    Theta=np.pi/90 #生成极坐标时的角度步长
```

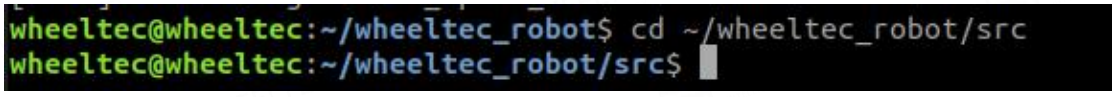
```
    CurveNumber=100 #极坐标交点上的曲线最低数量要求
    minLineLength = 100 #长度低于 100 像素点的直线被忽略
    maxLineGap = 10 #距离大于 10 像素点的点被认为在两条不同的直线上
    lines = cv2.HoughLinesP(image_threshold, Rho, Theta, CurveNumber,
minLineLength, maxLineGap)

    logest_line=0
    logest_lenghth=0
    try:
     #在识别出来的多条直线中进行比较，获得长度最长的直线
     logest_line=lines[0]
     for line in lines:
      for x1,y1,x2,y2 in line:
       lenghth=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)
      for x1,y1,x2,y2 in logest_line:
       logest_lenghth=(x1-x2)*(x1-x2)+(y1-y2)*(y1-y2)
      if lenghth>=logest_lenghth:
       logest_line=line
     print("Line detected.")
     print("Longes line's length is "+str(logest_lenghth**0.5)) #打印最长直线的
长度

     #判断最长直线在画面中是偏左还是偏右
     for x1,y1,x2,y2 in logest_line:
        cv2.line(cv_image, (x1,y1), (x2,y2), (255,0,0), 2) #把最长直线在画面中
标示出来
        X=int((x1+x2)/2)-25 #求直线中点坐标，用于判断直线偏移方向与程度。25 是
摄像头位置补偿参数，我们目前使用摄像头是双目摄像头，而 RGB 摄像头位于右方，不是位
于正中间
        Y=int((y1+y2)/2)
        cv2.circle(cv_image, (X, Y), 75, (0, 0, 255), 5)#在原图画圆，把直线中点
标示出来
        cv2.putText(cv_image,str(X),#把直线中点坐标用文字显示出来
                    (X-15, Y),#字体坐标
                    cv2.FONT_HERSHEY_SIMPLEX,
                    1,#字体大小
                    (0, 255, 0),#字体颜色 BGR
                    3)

        #求最长直线偏移程度，并根据偏移程度发布速度控制命令
        Turn_error=cols/2-X #直线偏左为正，偏右为负
        if Turn_error<30 and Turn_error>-30: #直线在中心一定范围内可以认为没有
偏移
           Turn_error=0
```

```
            self.twist.linear.x  = 0.2 #速度控制命令的前进速度大小0.2m/s
            self.twist.angular.z = Turn_error*0.0016 #速度控制命令的转向速度大小取决于直线偏移程度，左转为正，右转为负
        except TypeError or UnboundLocalError:
                    print("No line detected.")
                    self.twist.linear.x  = 0 #没有识别到直线则发布控制速度0，小车停止
                    self.twist.angular.z = 0

        #发布控制命令
        self.cmd_vel_pub.publish(self.twist)

        #直接使用OpenCV创建窗口显示图片
        cv2.imshow("cv_image", cv_image) #OpenCV创建窗口显示原图
        cv2.imshow("image_threshold", image_threshold) #OpenCV创建窗口显示最终处理图片
        #cv2.imshow("THRESHOLD",canvas) #显示动态调参窗口
        cv2.waitKey(1) #功能为刷新图像，若要创建窗口显示图片必须有这一函数。入口参数为延时时间，单位ms，为0则无限延时。函数返回延时时间内键盘按键的ASCII码值

if __name__ == '__main__': #这段判断的作用是，如果本py文件是直接运行的则判断通过执行if内的内容，如果是import到其他的py文件中被调用(模块重用)则判断不通过
  rospy.init_node("OpenCVLineFollow") #创建节点
  rospy.loginfo("OpenCVLineFollow node started") #打印ROS消息说明节点已开始运行
  Line_Follow() #直接运行image_converter()函数创建类，该类在运行期间会一直存在。因为该类没有需要调用的函数，所以使用赋值的形式：a=image_converter()
  rospy.spin() #相当于while(1),当订阅者接收到新消息时调用回调函数
```

6.4 创建巡线功能包并使用

① 创建功能包

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



catkin_create_pkg opencv_line_follower std_msgs rospy roscpp #在src文件夹下创建名为【opencv_line_follower】的功能包，后面std_msgs、rospy、roscpp为一些基本依赖

```
wheeltec@wheeltec:~/wheeltec_robot/src$ catkin_create_pkg opencv_line_follower s
td_msgs rospy roscpp
Created file opencv_line_follower/package.xml
Created file opencv_line_follower/CMakeLists.txt
Created folder opencv_line_follower/include/opencv_line_follower
Created folder opencv_line_follower/src
Successfully created files in /home/wheeltec/wheeltec_robot/src/opencv_line_foll
ower. Please adjust the values in package.xml.
wheeltec@wheeltec:~/wheeltec_robot/src$
```

cd opencv_line_follower #打开功能包文件夹

mkdir scripts #在功能包文件夹下创建【scripts】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【scripts】文件夹

```
wheeltec@wheeltec:~/wheeltec_robot/src$ cd opencv_line_follower
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower$ mkdir scripts
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower$ ls
CMakeLists.txt  include  package.xml  scripts  src
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower$
```

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

cd scripts/ #打开【scripts】文件夹

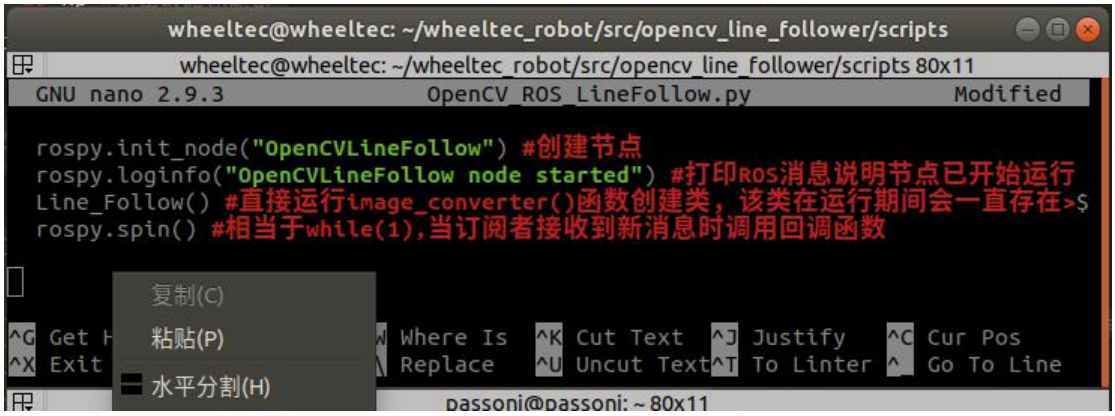
touch OpenCV_ROS_LineFollow.py #创建文件【OpenCV_ROS_LineFollow.py】

```
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower$ cd scripts/
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower/scripts$ touch OpenC
V_ROS_LineFollow.py
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower/scripts$ ls
OpenCV_ROS_LineFollow.py
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower/scripts$
```

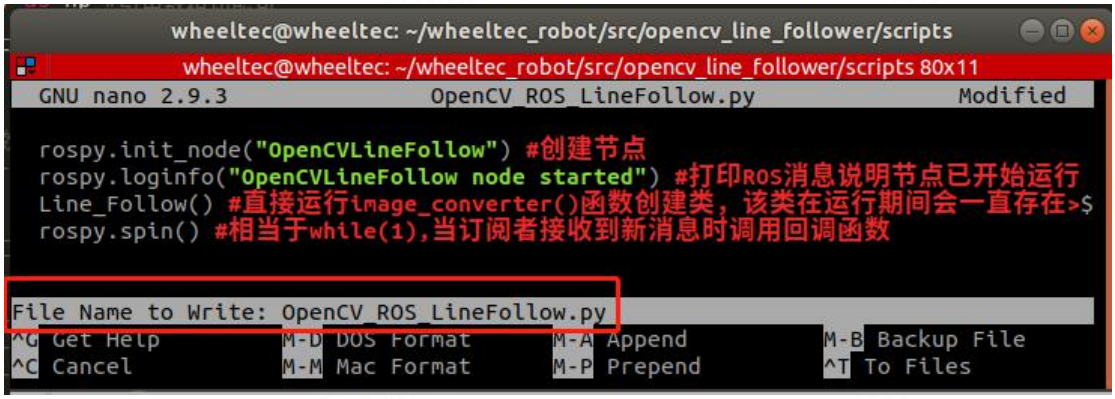
nano OpenCV_ROS_LineFollow.py #使用 nano 编辑器编辑文件

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

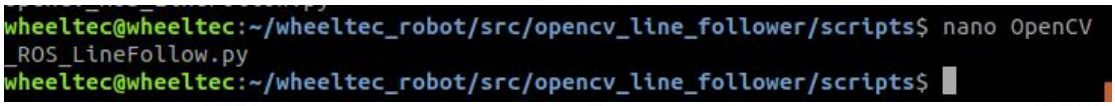
复制我们的例程程序内容，在 nano 编辑器单击右键，选择粘贴。



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



再按下同时 Ctrl+X 键，退出 nano 编辑器。



sudo chmod 777 OpenCV_ROS_LineFollow.py #给予【OpenCV_ROS_LineFollow.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 opencv_line_follower.launch #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【opencv_line_follower.launch】文件的全部内容。

```
<launch>

  <!-- 开启 RGB 摄像头节点  在 usb_cam 功能包下找到 usb_cam-test.launch 文件并运行
-->
  <include file="$(find usb_cam)/launch/usb_cam-test.launch" />

  <!-- 开启巡线节点  在 opencv_line_follower 功能包下找到
OpenCV_ROS_LineFollow.py 文件并运行，通过该文件创建的节点命名为
OpenCVLineFollow-->
  <node name='OpenCVLineFollow' pkg="opencv_line_follower"
type="OpenCV_ROS_LineFollow.py">
  </node>

  <!-- 开启机器人底层相关节点  在 turn_on_wheeltec_robot 功能包下找到
turn_on_wheeltec_robot.launch 文件并运行-->
  <include file="$(find
turn_on_wheeltec_robot)/launch/turn_on_wheeltec_robot.launch" />

</launch>
```

③ 编译并运行

catkin_make #返回工作空间根目录后输入【catkin_make】命令进行编译

```
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower/launch$ cd ..
wheeltec@wheeltec:~/wheeltec_robot/src/opencv_line_follower$ cd ..
wheeltec@wheeltec:~/wheeltec_robot/src$ cd ..
wheeltec@wheeltec:~/wheeltec_robot$ catkin_make
```

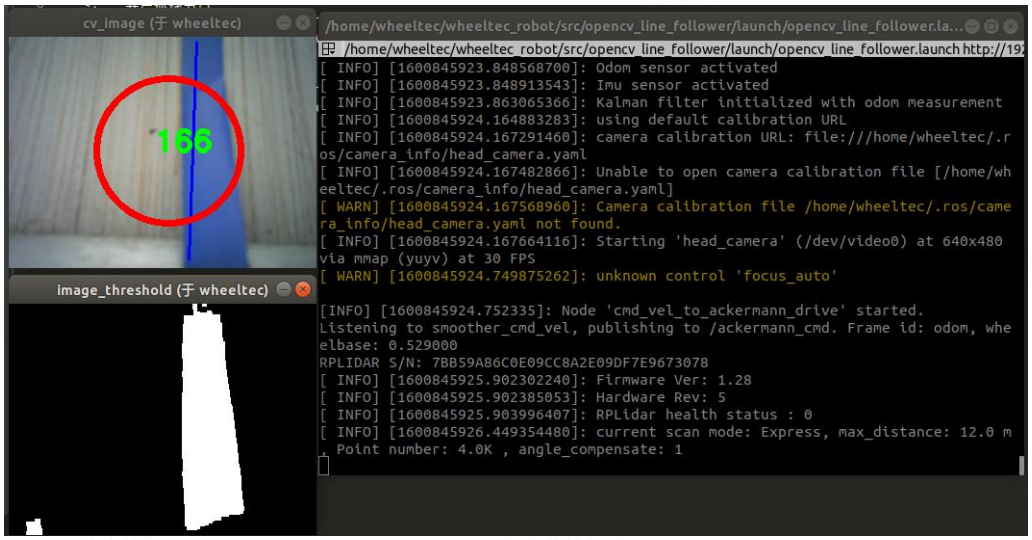
source devel/setup.bash #编译完成后输入命令 source 文件 setup.bash

```
wheeltec@wheeltec: ~/wheeltec_robot
wheeltec@wheeltec: ~/wheeltec_robot 80x11
make[2]: warning:  Clock skew detected.  Your build may be incomplete.
[ 94%] Built target planner
[ 94%] Built target move_base
make[2]: Warning: File '/home/wheeltec/wheeltec_robot/devel/lib/move_base/move_b
ase' has modification time 90925 s in the future
make[2]: warning:  Clock skew detected.  Your build may be incomplete.
[ 97%] Built target teb_local_planner
[ 98%] Built target move_base_node
[100%] Built target test_optim_node
wheeltec@wheeltec:~/wheeltec_robot$ source devel/setup.bash
wheeltec@wheeltec:~/wheeltec_robot$
```

roslaunch opencv_line_follower opencv_line_follower.launch #输入命令启动 launch 文件，运行程序。

```
wheeltec@wheeltec:~/wheeltec_robot$ roslaunch opencv_line_follower opencv_line_f
ollower.launch
```

运行成功。



④ 整体效果

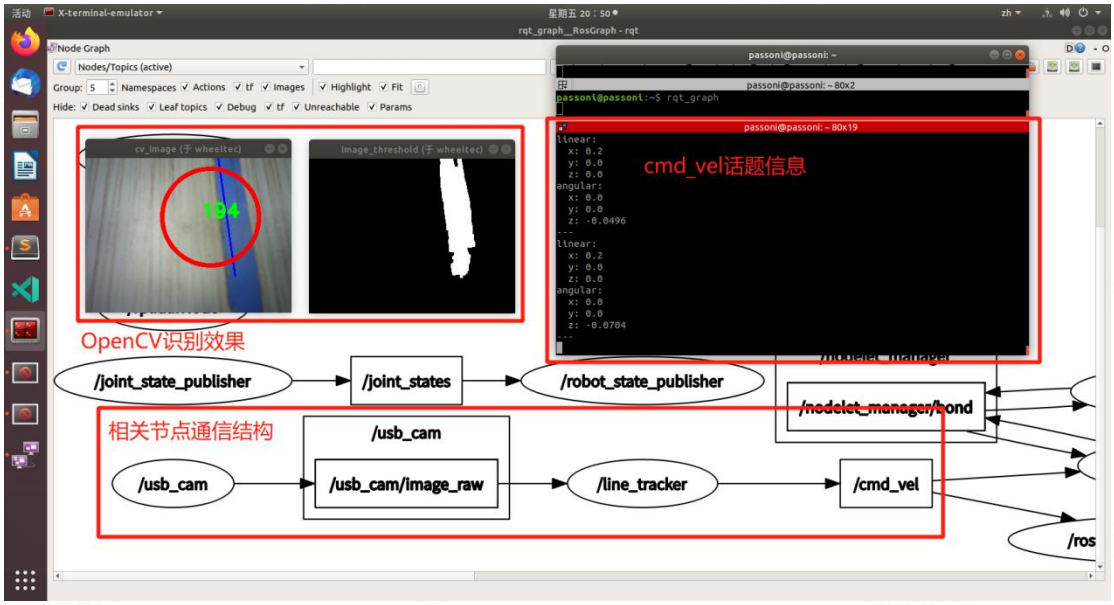


图 6-4-1 运行程序整体效果图

7. 在 ROS 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 ROS 参数服务器功能。

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

下图 5-1-1 为本章程序的执行流程图，重点在于寻找轮廓与轮廓区域内的深度信息处理。

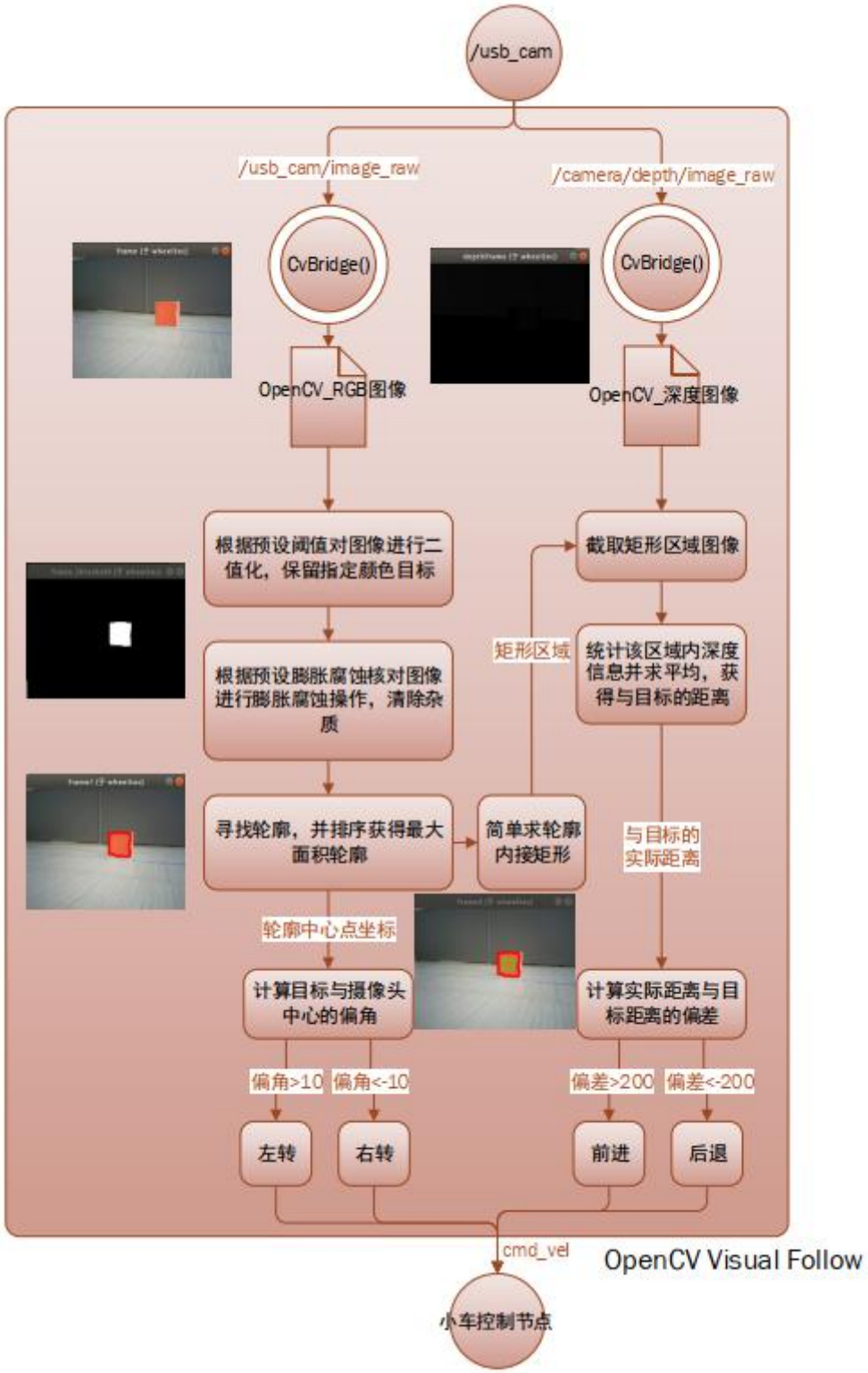


图 7-1-1 例程程序执行流程图

7.2 OpenCV 程序及解析

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

```
#!/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 rospy #引用ROS的Python接口功能包
import message_filters
#https://blog.csdn.net/chishuideyu/article/details/77479758
import cv2, cv_bridge #引用opencv功能包。cv_bridge是ROS图像消息和OpenCV图像之间转换的功能包
import numpy as np #引用数组功能包
from sensor_msgs.msg import Image #引用ROS内的图片消息格式
from sensor_msgs.msg import CompressedImage #引用ROS内的图片消息格式
from geometry_msgs.msg import Twist #引用ROS的速度消息格式

#定义视觉跟踪类
class VisualFollower:
    def __init__(self):#类初始化
                #从参数服务器获取相关参数，这些参数在launch文件中定义
                self.vertAngle=rospy.get_param('~visual_angle/vertical') #定义摄像头垂直可视角度大小
                self.horizontalAngle=rospy.get_param('~visual_angle/horizontal') #定义摄像头水平可视角度大小
                self.targetDistance=rospy.get_param('~targetDistance') #定义目标距离

self.velocity_foward_multiple=rospy.get_param('~velocity_multiple/forward') #前进后退速度放大倍率

self.velocity_turn_multiple=rospy.get_param('~velocity_multiple/turn')      #转向速度发大倍率

self.velocity_foward_restrict=rospy.get_param('~velocity_restrict/forward') #前进后退速度限幅

self.velocity_turn_restrict=rospy.get_param('~velocity_restrict/turn')      #转向速度限幅

                self.twist = Twist() #创建速度控制命令变量
                self.bridge = cv_bridge.CvBridge() #OpenCV与ROS的消息转换类

                #message_filters的作用是把订阅的数据同步后再在message_filters
```

```
的回调函数中使用
                im_sub = message_filters.Subscriber('/usb_cam/image_raw', Image)
                dep_sub = message_filters.Subscriber('/camera/depth/image_raw',
Image)
                self.timeSynchronizer =
message_filters.ApproximateTimeSynchronizer([im_sub, dep_sub], 10, 0.5)
                self.timeSynchronizer.registerCallback(self.VisualFollow)

                self.cmd_vel_pub = rospy.Publisher("cmd_vel", Twist, queue_size=1)
#速度控制命令发布者

    #视觉跟踪实现函数
    def VisualFollow(self, image_data, depth_data):
                #图像转换与预处理
                frame = self.bridge.imgmsg_to_cv2(image_data,
desired_encoding='bgr8')            #ROS 图像转 OpenCV 图像
                depthFrame = self.bridge.imgmsg_to_cv2(depth_data,
desired_encoding='passthrough')#ROS 图像转 OpenCV 图像
                frame = cv2.resize(frame, (320,240), interpolation=cv2.INTER_AREA)
#降低图像分辨率，以提高程序运行速度
                depthFrame = cv2.resize(depthFrame, (320,240),
interpolation=cv2.INTER_AREA)  #降低图像分辨率，以提高程序运行速度
                frame_height, frame_width, frame_channels = frame.shape #获得图
像尺寸高度、宽度、通道数
                #cv2.imshow("frame", frame)
                #cv2.imshow("depthFrame", depthFrame)

                #图像二值化
                targetUpper=np.array([112, 131, 252]) #红色
                targetLower=np.array([0, 0, 148])     #红色
                frame_threshold = cv2.inRange(frame, targetLower, targetUpper)

                #膨胀腐蚀
                kernel_width=5;
                kernel_height=5;
                kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_width,
kernel_height))
                frame_threshold = cv2.erode(frame_threshold, kernel)
                frame_threshold = cv2.dilate(frame_threshold, kernel)
                frame_threshold = cv2.dilate(frame_threshold, kernel)
                frame_threshold = cv2.erode(frame_threshold, kernel)
                #cv2.imshow("frame_threshold", frame_threshold) #显示预处理结果

                #寻找轮廓 API，功能：输入图片，返回原图、轮廓对象、轮廓父子结构
```

```
                img, contours, hierarchy =
cv2.findContours(frame_threshold.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE) #cv2.RETR_EXTERNAL: 只检测外轮廓,
cv2.CHAIN_APPROX_SIMPLE): 只保留轮廓的交点

                #如果找到了轮廓,则保留最大面积轮廓—>规划一个比轮廓形状小的区域
—>获得该区域的深度信息(即与目标的距离)—>从深度信息、区域中心坐标判断并给 twist
赋值以进行前进后退左右转
                try:
                 contours_sorted=sorted(contours, key=cv2.contourArea,
reverse=True) #按照轮廓面积进行排序
                 cv2.drawContours(frame, contours_sorted, 0, (0,0,255), 3)
#画出第一个轮廓, 因为已排序, 所以即为最大的轮廓

                 (x, y),radius = cv2.minEnclosingCircle(contours_sorted[0]) #对
最大轮廓求最小外接圆, 并返回中心坐标(浮点型), 半径(浮点型)
                 x_float=x #保留浮点型, 用于后面计算角度
                 y_float=y #保留浮点型, 用于后面计算角度
                 x=int(x) #转为整型才能用于图像处理
                 y=int(y) #转为整型才能用于图像处理
                 radius=int(radius) #转为整型才能用于图像处理
                 #cv2.imshow("frame1", frame) #显示识别效果
                 cv2.circle(frame, (x, y), radius, (0, 255, 255), 3) #画出最小外
接圆
                 target_radius=radius/3 #用于简单求轮廓内接矩形
                 depthObject = depthFrame[(y-target_radius):(y+target_radius),
(x-target_radius):(x+target_radius)]#截取目标区域的深度图, 用于取区域内深度信息,
frame[(height_start:height_end), (width_start:height_end)]

cv2.rectangle(frame,(x-target_radius,y-target_radius),(x+target_radius,
y+target_radius), (0,255,0), 2)#画出轮廓内接矩形
                 depthArray = depthObject[~np.isnan(depthObject)]#获取区域内非空
的深度信息
                 averageDistance = np.mean(depthArray)          #求平均得到与目
标的距离, 单位:毫米

                 #根据区域中心点坐标与摄像头可视角度计算目标与摄像头中心的偏角
                 angleX=self.horizontalAngle*(0.5-x_float/frame_width)
                 angleY=self.vertAngle*(0.5-y_float/frame_height)

                 #打印偏角、距离信息
                 rospy.loginfo("angleX:"+str(angleX))
                 rospy.loginfo("angleY:"+str(angleY))
                 rospy.loginfo("distance:"+str(averageDistance))
```

```
                #深度摄像头在距离太近时是直接没有深度信息的，如果平均距离小于1
则判断深度信息错误，不发布速度控制命令
                if(averageDistance>1):
                 Distance_Bias=averageDistance-self.targetDistance #求实际距离
与目标距离的偏差

                 #偏差较小时不进行移动
                 if(angleX<10 and angleX>-10):
                  angleX=0
                 if (Distance_Bias<200 and Distance_Bias>-200):
                  Distance_Bias=0

                 #根据偏差大小给速度控制命令赋值
                 self.twist.linear.x  =
Distance_Bias/1000*self.velocity_foward_multiple #前进为正，后退为负，单位：米
                 self.twist.angular.z =
angleX/180*3.14*self.velocity_turn_multiple      #左转为正，右转为负，单位：弧度

                 #前进后退速度限幅
                 if   (self.twist.linear.x> self.velocity_foward_restrict):
                   self.twist.linear.x= self.velocity_foward_restrict
                 elif (self.twist.linear.x<-self.velocity_foward_restrict):
                   self.twist.linear.x=-self.velocity_foward_restrict

                 #转向速度限幅
                 if   (self.twist.linear.z> self.velocity_turn_restrict):
                   self.twist.linear.x= self.velocity_foward_restrict
                 elif (self.twist.linear.z<-self.velocity_turn_restrict):
                   self.twist.linear.x=-self.velocity_turn_restrict

               #没有找到轮廓，速度控制命令置零
               except IndexError:
                rospy.logwarn("contours not found")
                self.twist.linear.x  = 0
                self.twist.angular.z = 0

               #发布控制命令
               self.cmd_vel_pub.publish(self.twist)
               cv2.imshow("frame2", frame) #显示识别效果
               cv2.waitKey(1)

if __name__ == '__main__': #这段判断的作用是，如果本 py 文件是直接运行的则判断通
过执行 if 内的内容，如果是 import 到其他的 py 文件中被调用(模块重用)则判断不通过
```

```
  rospy.init_node("opencv") #创建节点
  rospy.loginfo("OpenCV VisualFollow node started") #打印ROS消息说明节点已开始运行
  Visual_follower=VisualFollower()
  rospy.spin() #相当于while(1),当订阅者接收到新消息时调用回调函数
```

7.3 launch 文件(使用参数服务器)

文件名为【opencv_visual_foloower.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 opencv_visual_foloower.launch

```
<launch>

  <!-- 开启RGB摄像头  -->
  <include file="$(find usb_cam)/launch/usb_cam-test.launch" />

  <!-- 开启深度摄像头  -->
  <include file="$(find astra_camera)/launch/astra.launch" />

  <!-- 开启色块跟踪节点  -->
  <node name='opencv_visual_follower' pkg="simple_follower" type="OpenCV_ROS_VisualFollow.py">

    <!--  上传相关参数到参数服务器 -->
    <rosparam ns='visual_angle'>
      vertical :   46.7
      horizontal : 60
    </rosparam>
    <rosparam ns='velocity_multiple'>
      forward: 0.5
      turn:    1.2
    </rosparam>
    <rosparam ns='velocity_restrict'>
      forward: 0.35
      turn:    2
    </rosparam>
    <param name='targetDistance' value='600' type='double' />

  </node>

  <!-- 开启机器人底层相关节点  -->
  <include file="$(find turn_on_wheeltec_robot)/launch/turn_on_wheeltec_robot.launch" />
```

```
</launch>
```

② 启动效果

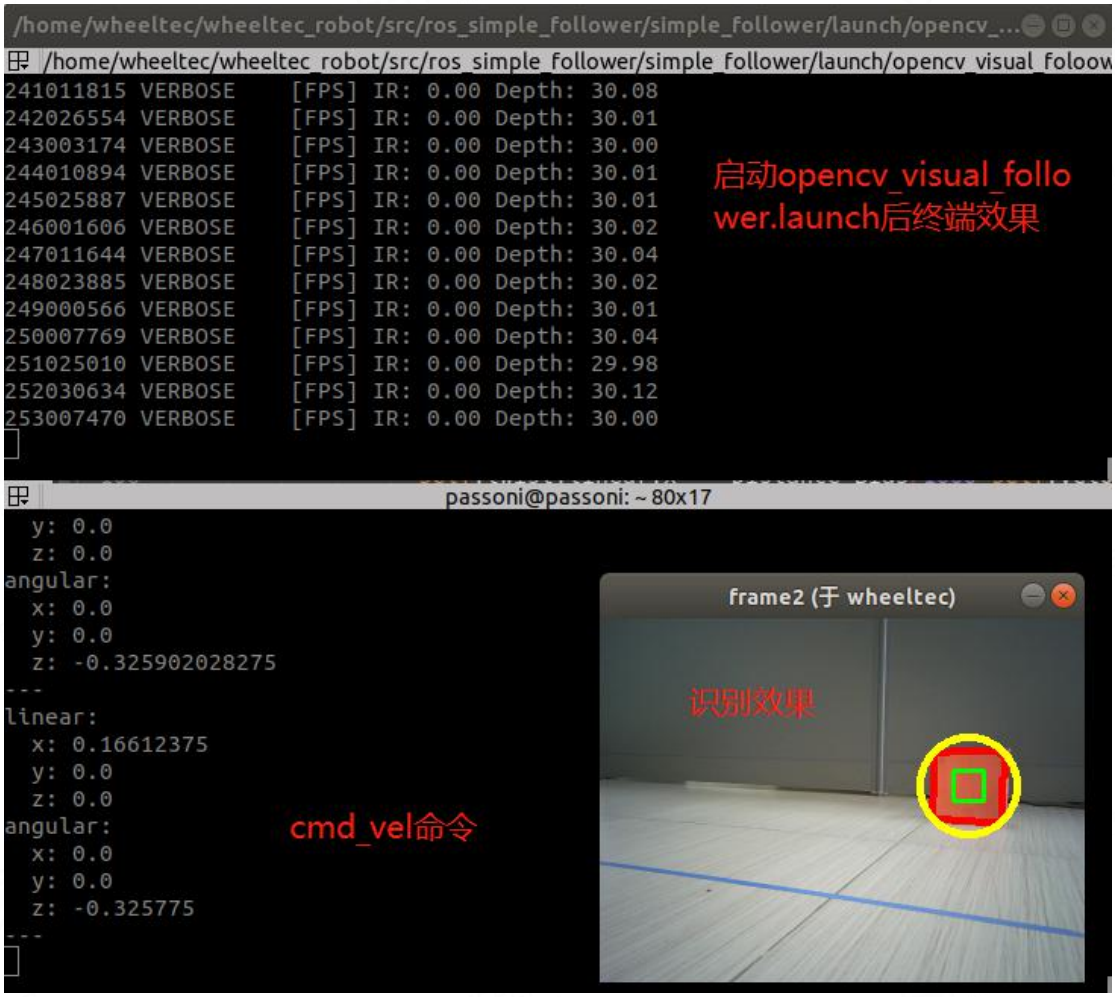


图 7-3-1 运行效果