Chapter20 真实场景下的小车行驶

实验场景

本次实验模拟在红绿灯路口处小车的行为：

- 颜色判断红灯停、绿灯行、黄灯减速；
- 在双实白线约束的车道内行驶；

代码部署

起初设计了多线程，主线程负责小车行驶，另外两个线程一个负责信号灯识别，一个负责车道识别。但部署到树莓派上时，发现树莓派好像不支持多线程调用opencv中的imshow、waitKey等函数（大概是版本问题），所以做出了两个版本。在物理机上可以使用多线程，在树莓派上就是一个巨长的流水线main。

多线程版本

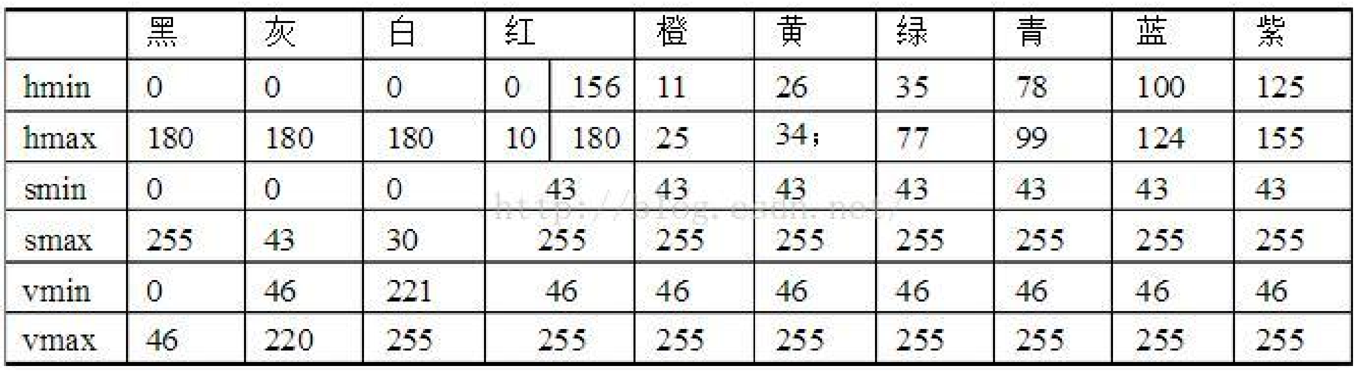
主线程

```
import cv2 as cv
import numpy as np
import math

import time
from color_interval import *
import threading
import track_line
import signal_light

#from tcp_control.car_run import *

def cnt_lock(lock):
    lock.acquire()
    signal_light.red_cnt = signal_light.green_cnt = signal_light.yellow_cnt = 0
    lock.release()


if __name__ == '__main__':
    cap=cv.VideoCapture(0)

    t1 = threading.Thread(target=track_line.track_line, args=[cap,"black"])   #可
选择颜色
    t2 = threading.Thread(target=signal_light.traffic_light, args=[cap])
    t1.start()
    t2.start()

    while 1:
        #run(LOW_SPEED,LOW_SPEED)
        if track_line.steer_angle>=0:
            #right(LOW_SPEED)
            pass
        else:
            #left(LOW_SPEED)
            pass

        if signal_light.red_cnt==signal_light.flag_1s:
            #brake()
            print("brake")
            cnt_lock(signal_light.threadLock)

        if signal_light.green_cnt==signal_light.flag_1s:
            #run(LOW_SPEED,LOW_SPEED)
            print("run")
            cnt_lock(signal_light.threadLock)
        if signal_light.yellow_cnt==signal_light.flag_1s:
            #减速
            cnt_lock(signal_light.threadLock)
            print("yellow")
            pass
```

巡线线程

精准颜色识别

之前说到，在实际场景中进行颜色识别时，一般采用hsv空间。hsv基础阈值表如下：



	黑	灰	白	红		橙	黄	绿	青	蓝	紫
hmin	0	0	0	0	156	11	26	35	78	100	125
hmax	180	180	180	10	180	25	34；	77	99	124	155
smin	0	0	0	43		43	43	43	43	43	43
smax	255	43	30	255		255	255	255	255	255	255
vmin	0	46	221	46		46	46	46	46	46	46
vmax	46	220	255	255		255	255	255	255	255	255

在color_interval.py中储存各种颜色的阈值，以方便调用：

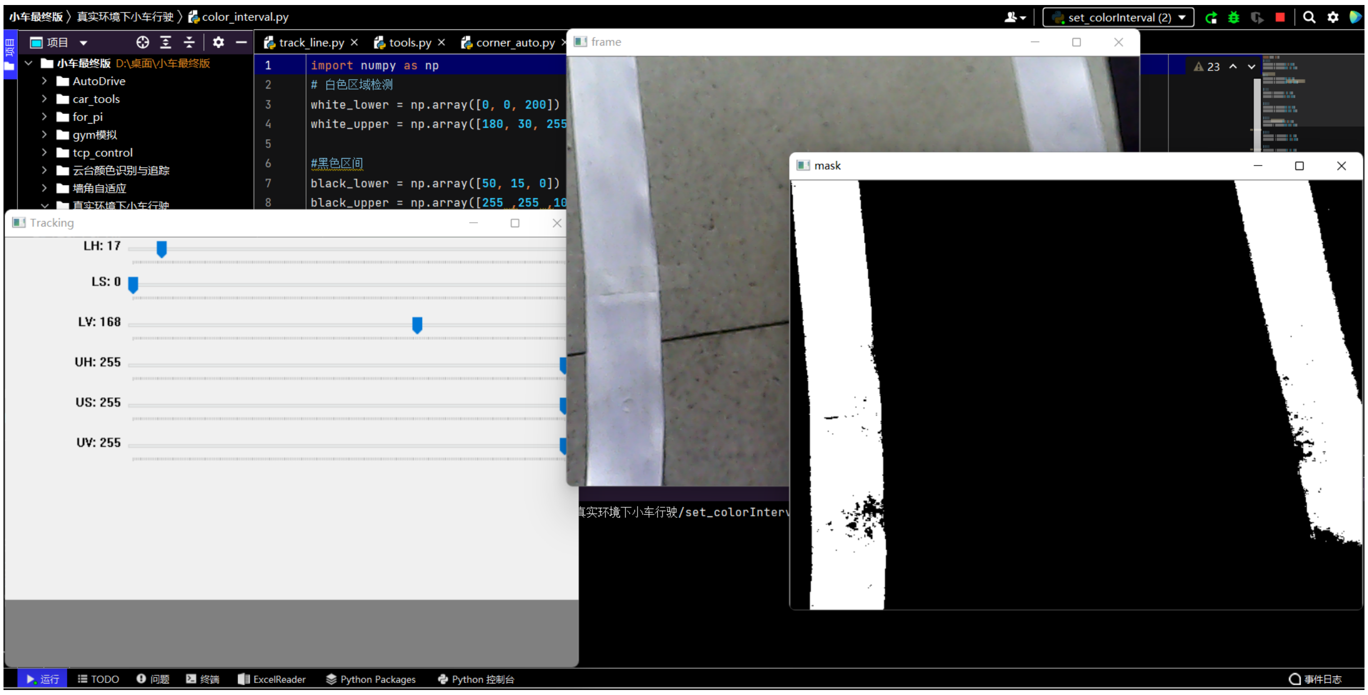
```
import numpy as np
# 白色区域检测
white_lower = np.array([0, 0, 200])
white_upper = np.array([180, 30, 255])

#黑色区间
black_lower = np.array([50, 15, 0])
black_upper = np.array([255 ,255 ,106])

# 红色区间
red_lower = np.array([0, 43, 46])
```

```
red_upper = np.array([10, 255, 255])

# #绿色区间
green_lower = np.array([35, 43, 46])
green_upper = np.array([77, 255, 255])

# #蓝色区间
blue_lower=np.array([100, 43, 46])
blue_upper = np.array([124, 255, 255])

# #黄色区间
yellow_lower = np.array([26, 43, 46])
yellow_upper = np.array([34, 255, 255])

# #橙色区间
orange_lower = np.array([11, 43, 46])
orange_upper = np.array([25, 255, 255])

color_dict={"red":[red_lower,red_upper],"green":[green_lower,green_upper],
            "blue":[blue_lower,blue_upper],"yellow":[yellow_lower,yellow_upper],
            "black":[black_lower,black_upper],"white":[white_lower,white_upper]}
```

但这是一个相当宽泛的范围，与我们实际想要的存在较大偏差，会捕捉到一些噪声。所以要精确地捕捉要追踪的线，就要打开摄像头进行调试。设计了一个可以精确确定出我们想要的阈值的程序，可以通过trackbar来调节阈值，观察掩膜来确定是否捕捉到想要颜色。通过调节tracebar来确定好六个阈值后，可以直接在color_interval.py中修改相应阈值。

```
#set_colorInterval.py
import cv2 as  cv
import numpy as np

def callback(x):
    pass

cv.namedWindow("Tracking")
cv.namedWindow("frame")

cv.createTrackbar("LH","Tracking",0,255,callback)
cv.createTrackbar("LS","Tracking",0,255,callback)
cv.createTrackbar("LV","Tracking",0,255,callback)
cv.createTrackbar("UH","Tracking",0,255,callback)
cv.createTrackbar("US","Tracking",0,255,callback)
cv.createTrackbar("UV","Tracking",0,255,callback)

cap=cv.VideoCapture(0)
while True:
    ret,frame=cap.read()
    #frame = cv.imread(r"color.jpg")
    #frame = cv.resize(frame,(480,480))
    if not ret:
        print("VideoCapture error")
```

```
            break
        hsv = cv.cvtColor(frame,cv.COLOR_BGR2HSV)

        l_h = cv.getTrackbarPos("LH","Tracking")
        l_s = cv.getTrackbarPos("LS","Tracking")
        l_v = cv.getTrackbarPos("LV","Tracking")
        u_h = cv.getTrackbarPos("UH","Tracking")
        u_s = cv.getTrackbarPos("US","Tracking")
        u_v = cv.getTrackbarPos("UV","Tracking")


        l_g = np.array([l_h, l_s, l_v])
        u_g = np.array([u_h,u_s,u_v])

        mask = cv.inRange(hsv,l_g,u_g)

        res=cv.bitwise_and(frame,frame,mask=mask)

        cv.imshow("frame", frame)
        cv.imshow("mask", mask)
        cv.imshow("res", res)
        key = cv.waitKey(1)
        if key == ord("q"):
            break

    cv.destroyAllWindows()
```



```
import numpy as np
# 白色区域检测
white_lower = np.array([0, 0, 200])
white_upper = np.array([180, 30, 255

#黑色区间
black_lower = np.array([50, 15, 0])
black_upper = np.array([255 ,255 ,10
```

LH: 17
LS: 0
LV: 168
UH: 255
US: 255
UV: 255

巡线线程

我们采用一个线程来实现巡线。

要实现主线程与巡线线程之间的信息传递，使用一个共享变量steer_angle来进行传递。巡线线程确定小车要转弯的角度，主线程对角度进行处理，然后驱动小车运动。当然，这里面要对steer_angle加锁。

```python
#track_line.py
import cv2 as cv
import numpy as np
import time
from color_interval import *
# 导入自定义库
from tools import region_of_interest, detect_line, average_lines, display_line

steer_angle =0

def trackbar_callback(value):
    pass

def track_line(cap,color:str):
    '''
    线程函数
    '''
    global steer_angle

    cv.namedWindow("mask", cv.WINDOW_AUTOSIZE)
    cv.namedWindow("edge", cv.WINDOW_AUTOSIZE)
    cv.namedWindow("black", cv.WINDOW_AUTOSIZE)

    cv.createTrackbar("low_threshold", "edge", 0, 1000, trackbar_callback)
    cv.createTrackbar("high_threshold", "edge", 0, 1000, trackbar_callback)

    color_lower,color_upper=color_dict[color]

    cnt = 0
    while 1:
        ret, frame = cap.read()
        if not ret:
            continue
        height, width, _ = frame.shape
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        mask = cv.inRange(hsv, color_lower, color_upper)
        mask = cv.erode(mask, None, iterations=2)
        # mask = cv.dilate(mask, None, iterations=1)
        # mask = cv.GaussianBlur(mask, (3, 3), 0)
        cv.imshow("mask",mask)
        # 边缘检测

        thresh1 =cv.getTrackbarPos("low_threshold","edge")
        thresh2 =cv.getTrackbarPos("high_threshold", "edge")

        edge = cv.Canny(mask, thresh1, thresh2)
        cv.imshow("edge",edge)

        # region of interest 获取感兴趣区域的边缘
        black_left_roi = region_of_interest(edge, color="left")
        black_right_roi = region_of_interest(edge, color="right")
        left_frame = frame.copy()
        right_frame = frame.copy()

        # 线段检测
        left_lines = detect_line(black_left_roi)
        right_lines = detect_line(black_right_roi)
        left_lane = average_lines(frame, left_lines, direction='left')
        right_lane = average_lines(frame, right_lines, direction="right")
        show = display_line(left_frame, left_lane, line_color=(255, 0,
0),line_width=20)
        show = display_line(show, right_lane, line_color=(0, 255,
0),line_width=20)

        cv.imshow("black", show)

        # 计算转向角
        x_offset = 0    #一般为几十
        y_offset = 0    #固定为window_height/2=240
        if len(left_lane) > 0 and len(right_lane) > 0:  # 检测到2条线
            _, _, left_x2, _ = left_lane[0][0]
            _, _, right_x2, _ = right_lane[0][0]
            mid = int(width / 2)
            x_offset = (left_x2 + right_x2) / 2 - mid
            y_offset = int(height / 2)
        elif len(left_lane) > 0 and len(left_lane[0]) == 1:  # 只检测到黄色行道线
            x1, _, x2, _ = left_lane[0][0]
            x_offset = x2 - x1
            y_offset = int(height / 2)
        elif len(right_lane) > 0 and len(right_lane[0]) == 1:  # 只检测到白色行道线
            x1, _, x2, _ = right_lane[0][0]
            x_offset = x2 - x1
            y_offset = int(height / 2)
        else:  # 一条线都没检测到
            print('检测不到行道线')
            # break

        print("x:", x_offset)
        print("y:", y_offset)
        if y_offset==0:
            continue
        steer_angle = x_offset*10 // y_offset
        print("steer_angle:",steer_angle)
        cnt += 1

        print("speed",30*(1+0.1*abs(steer_angle)))

        key = cv.waitKey(1)
        if key == ord("q"):
            break
        elif key == ord("c"):
            cv.imwrite("cut" + str(cnt) + ".jpg", show)

    cv.destroyAllWindows()
```

```
if __name__=="__main__":
    pass
```

信号灯捕捉线程

通过set_colorInterval.py我们可以确定当前场景下巡线、红绿黄灯的阈值。

现在来进行红绿黄的捕捉。

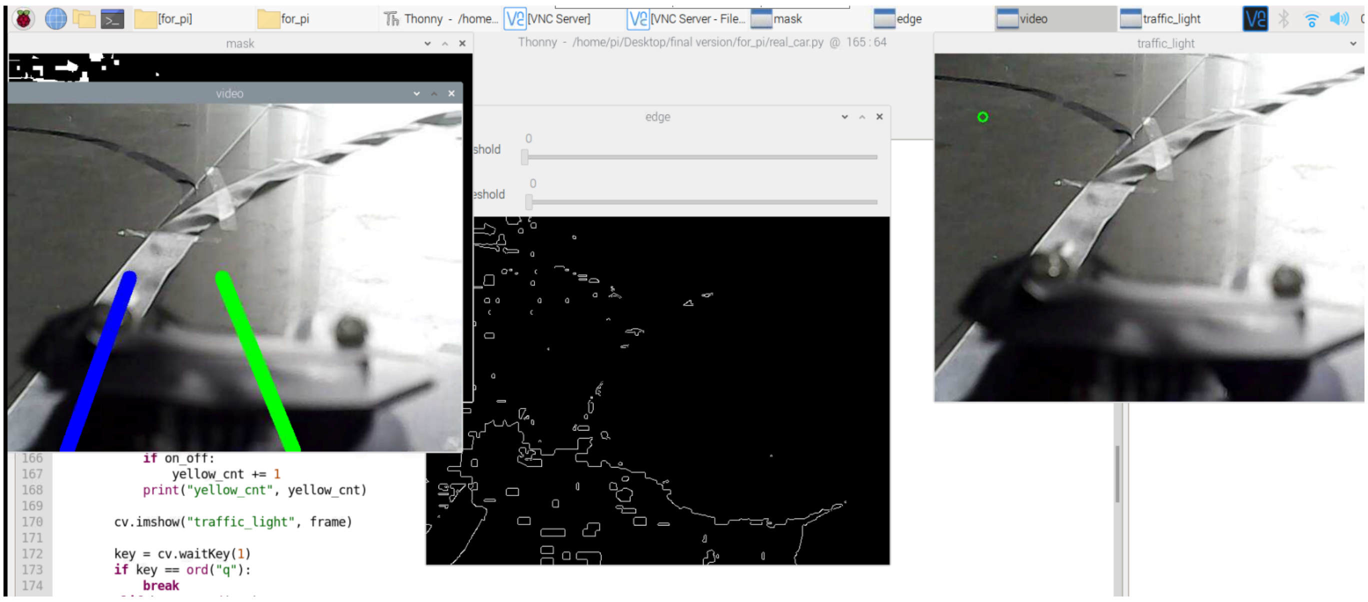
不妨将一个大的色块认为是信号灯（简单期间）。对于同时出现多个相同色块的情况，选取面积最大的一个。对于同时出现多个颜色的色块的情况，选取进行霍夫圆检测后的半径最大的色块。同样使用一个线程来实现，并使用四个共享变量来与主线程沟通（要加锁）：red_cnt，green_cnt，yellow_cnt，flag_1s。这四个变量用于统计捕捉到各种颜色的帧数，当帧数达到flag_1s后，可以认为该信号灯有效。

```
#signal_light.py

import cv2 as cv
import numpy as np
from color_interval import *
import threading
red_cnt=0
green_cnt=0
yellow_cnt=0
flag_1s=10
#连续亮灯到达一秒 才算是识别到 但由于算力的问题 需要修改

threadLock = threading.Lock()

def find_contours(hsv_frame,color:str):
    '''
    传入hsv帧和选定颜色，在帧中找出所有边缘，并选出最大的的边缘。
    然后使用霍夫圆检测，确定最大色块的外接圆的半径以及圆心。
    返回四个值：色块圆心x，y 半径 边缘
    '''
    color_lower,color_upper=color_dict[color]
    mask = cv.inRange(hsv_frame, color_lower, color_upper)
    mask = cv.erode(mask, None, iterations=2)
    mask = cv.dilate(mask, None, iterations=2)
    mask = cv.GaussianBlur(mask, (3, 3), 0)
    contours = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)[-2]
    if len(contours)==0:
        return 0,0,0,0
    contour = max(contours, key=cv.contourArea)
    (x,y), radius = cv.minEnclosingCircle(contour)
    return x,y,radius,contour

def traffic_light(cap):
    '''
    线程函数
    '''
    global red_cnt,green_cnt,yellow_cnt

    cv.namedWindow("traffic_light",cv.WINDOW_AUTOSIZE)
    while 1:
        ret,frame=cap.read()
        if not ret:
            print("in traffic_light thread,video read error!")
            break
        frame = cv.GaussianBlur(frame, (5, 5), 0)
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)

        red_x,red_y,red_radius,red_contour=find_contours(hsv,"red")
        green_x,green_y,green_radius,green_contour=find_contours(hsv,"green")
        yellow_x,yellow_y,yellow_radius,yellow_contour=find_contours(hsv,"yellow")

        max_radius=max(red_radius,green_radius,yellow_radius)
        #对于不同颜色的色块，选择半径最大的。
        if max_radius<50:     #半径过小的直接忽略。
            pass
        if max_radius==red_radius:
            max_x,max_y=red_x,red_y
            cv.circle(frame, (int(max_x),int(max_y)), int(max_radius),(0,0,255),2)

            threadLock.acquire()
            red_cnt += 1
            #print("red_cnt", red_cnt)
            threadLock.release()


        elif max_radius==green_radius:
            max_x, max_y = green_x, green_y
            cv.circle(frame, (int(max_x),int(max_y)), int(max_radius), (0, 255,
0), 2)

            threadLock.acquire()
            green_cnt += 1
            #print("green_cnt", green_cnt)
            threadLock.release()
        else:
            max_x, max_y = yellow_x, yellow_y
            cv.circle(frame, (int(max_x),int(max_y)), int(max_radius), (0, 255,
255), 2)

            threadLock.acquire()
            yellow_cnt += 1
            #print("yellow_cnt", yellow_cnt)
            threadLock.release()

        cv.imshow("traffic_light",frame)
        key=cv.waitKey(1)
        if key==ord("q"):
            break
```

```
if __name__=="__main__":
    cap=cv.VideoCapture(0)
    threadLock = threading.Lock()
    while 1:
        ret,frame=cap.read()
        cv.imshow("raw_video",frame)
        if not ret:
            continue
        frame=traffic_light(frame)
        cv.imshow("signal_light",frame)
        if cv.waitKey(1000//30)==ord("q"):
            break
```

树莓派版本

```
import cv2 as cv
import RPi.GPIO as GPIO
import time

from car_tools.key import *
from color_interval import *
from tools import *
from car_tools.car_run import *


red_cnt=0
green_cnt=0
yellow_cnt=0
flag_1s=30
#连续亮灯到达一秒 才算是识别到 但由于算力的问题 需要修改



on_off=0    #按键是否按下
key_pressed_first=0
steer_angle = 0


def key_pressed_callback(pin):
    global on_off,key_pressed_first
    if not on_off:
        on_off = 1
        time.sleep(0.5)
    else:
        brake()
        exit(0)

def find_contours(hsv_frame,color:str):
    color_lower,color_upper=color_dict[color]
    mask = cv.inRange(hsv_frame, color_lower, color_upper)
    mask = cv.erode(mask, None, iterations=1)
    #mask = cv.dilate(mask, None, iterations=2)
    #mask = cv.GaussianBlur(mask, (3, 3), 0)
    contours = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)[-2]
    if len(contours)==0:
        return 0,0,0,0
    contour = max(contours, key=cv.contourArea)
    (x,y), radius = cv.minEnclosingCircle(contour)
    return x,y,radius,contour

def trackbar_callback(value):
    pass

if __name__ == '__main__':
    key_init()
    car_run_init()
    GPIO.add_event_detect(key, GPIO.RISING, key_pressed_callback, bouncetime=15)
    cap=cv.VideoCapture(0)

    cv.namedWindow("mask", cv.WINDOW_AUTOSIZE)
    cv.namedWindow("edge", cv.WINDOW_AUTOSIZE)
    cv.namedWindow("video", cv.WINDOW_AUTOSIZE)
    cv.namedWindow("traffic_light", cv.WINDOW_AUTOSIZE)

    #canny的阈值
    cv.createTrackbar("low_threshold", "edge", 0, 1000, trackbar_callback)
    cv.createTrackbar("high_threshold", "edge", 0, 1000, trackbar_callback)

    color_lower, color_upper = color_dict["black"]

    cnt = 0   #截图序号


    while 1:
        ret, frame = cap.read()
        if not ret:
            print("video read error!")
            break

        '''
        track line
        '''
        height, width, _ = frame.shape
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        mask = cv.inRange(hsv, color_lower, color_upper)
        mask = cv.erode(mask, None, iterations=1)
        # mask = cv.dilate(mask, None, iterations=1)
        # mask = cv.GaussianBlur(mask, (3, 3), 0)
        cv.imshow("mask", mask)

        # 边缘检测
        thresh1 = cv.getTrackbarPos("low_threshold", "edge")
        thresh2 = cv.getTrackbarPos("high_threshold", "edge")

        edge = cv.Canny(mask, thresh1, thresh2)
        cv.imshow("edge", edge)

        # region of interest 获取感兴趣区域的边缘
        black_left_roi = region_of_interest(edge, color="left")
        black_right_roi = region_of_interest(edge, color="right")

        left_frame = frame.copy()
        right_frame = frame.copy()

        # 线段检测
        left_lines = detect_line(black_left_roi)
        right_lines = detect_line(black_right_roi)
        left_lane = average_lines(frame, left_lines, direction='left')
        right_lane = average_lines(frame, right_lines, direction="right")
        show = display_line(left_frame, left_lane, line_color=(255, 0, 0),
line_width=20)
        show = display_line(show, right_lane, line_color=(0, 255, 0),
line_width=20)
        cv.imshow("video", show)

        # 计算转向角
        x_offset = 0
        y_offset = 0
        if len(left_lane) > 0 and len(right_lane) > 0:  # 检测到2条线
            _, _, left_x2, _ = left_lane[0][0]
            _, _, right_x2, _ = right_lane[0][0]
            mid = int(width / 2)
            x_offset = (left_x2 + right_x2) / 2 - mid
            y_offset = int(height / 2)
        elif len(left_lane) > 0 and len(left_lane[0]) == 1:  # 只检测到黄色行道线
            x1, _, x2, _ = left_lane[0][0]
            x_offset = x2 - x1
            y_offset = int(height / 2)
        elif len(right_lane) > 0 and len(right_lane[0]) == 1:  # 只检测到白色行道线
            x1, _, x2, _ = right_lane[0][0]
            x_offset = x2 - x1
            y_offset = int(height / 2)
        else:  # 一条线都没检测到
            print('检测不到行道线')

        # print("x:", x_offset)
        # print("y:", y_offset)
        if y_offset == 0:
            continue
        steer_angle = x_offset * 10// y_offset
        turn_speed=30*(1+0.1*steer_angle)
        if turn_speed>80:
            turn_speed=80
        '''
        signal light
        '''
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)

        red_x, red_y, red_radius, red_contour = find_contours(hsv, "red")
        green_x, green_y, green_radius, green_contour = find_contours(hsv,
"green")
        yellow_x, yellow_y, yellow_radius, yellow_contour = find_contours(hsv,
"yellow")

        max_radius = max(red_radius, green_radius, yellow_radius)
        if max_radius < 50:
            pass
        if max_radius == red_radius:
            max_x, max_y = red_x, red_y
            cv.circle(frame, (int(max_x), int(max_y)), int(max_radius), (0, 0,
255), 2)
            if on_off:
                red_cnt += 1
            print("red_cnt", red_cnt)

        elif max_radius == green_radius:
            max_x, max_y = green_x, green_y
            cv.circle(frame, (int(max_x), int(max_y)), int(max_radius), (0, 255,
0), 2)
            if on_off:
                green_cnt += 1
            print("green_cnt", green_cnt)

        else:
            max_x, max_y = yellow_x, yellow_y
            cv.circle(frame, (int(max_x), int(max_y)), int(max_radius), (0, 255,
255), 2)
            if on_off:
                yellow_cnt += 1
            print("yellow_cnt", yellow_cnt)

        cv.imshow("traffic_light", frame)

        key = cv.waitKey(1)
        if key == ord("q"):
            break
        elif key == ord("c"):
            cv.imwrite("cut" + str(cnt) + ".jpg", show)
        cnt += 1

        '''
        小车运动
        '''
        if on_off:
            if key_pressed_first==0:
                time.sleep(5)
                key_pressed_first=1

            run(10,10)
            if steer_angle>=0:
                right(turn_speed)
                time.sleep(0.1)
                run()
            else:
                left(turn_speed)
                time.sleep(0.1)

            if red_cnt==flag_1s:
                brake()
                red_cnt=0
                green_cnt=0
                yellow_cnt=0
                print("brake")

            if green_cnt==flag_1s:
                run(10,10)
                red_cnt=0
                green_cnt=0
                yellow_cnt=0
                time.sleep(0.05)
                print("run")

            if yellow_cnt==flag_1s:
                #减速
                red_cnt=0
                green_cnt=0
                yellow_cnt=0
                curr_speed=10
                while curr_speed>=0:
                    curr_speed-=5
                    run(curr_speed,curr_speed)
                    time.sleep(1)
                    print("slow down")
```



```
        if on_off:
            yellow_cnt += 1
        print("yellow_cnt", yellow_cnt)

    cv.imshow("traffic_light", frame)

    key = cv.waitKey(1)
    if key == ord("q"):
        break
```

实验成果

本次实验模拟了小车在红绿灯路口处的行为：

- 颜色判断红灯停、绿灯行、黄灯减速；
- 在双实白线约束的车道内行驶；

但需要注意的是，要尽可能避免环境噪声对摄像头捕捉的影响，可以选用粉色或绿色等彩带来进行模拟。