上海交通大學

课程设计报告

课程名称：《工程实践与科技创新 3F》

设计题目：基于计算机视觉的平台实验仿真系统实践

实验时间：2023 年 2 月 13 日 至 2023 年 5 月 29 日

学院(系)：电子信息与电气工程学院

专　　业：自动化

学生姓名：李厚霖　学号：520020910007

2023 年　6 月　18 日

目录

1.实验任务介绍

我们需要在基于计算机视觉的无人驾驶平台实验仿真系统上实现仿真小车在道路场景、停车场中的自动驾驶，该任务要求我们能够熟练地运用数字图像处理与简单控制的相关知识，针对不同场景、不同状况实现仿真小车的自动驾驶。

该部分分为两个任务，分别为地下停车场停车任务和城市街道自动驾驶任务。

该平台的场景和交互窗口如下图所示：

图 2 平台场景设计

图 3 平台交互窗口展示

2.基于计算机视觉的无人驾驶平台实验仿真系统的实践

基于计算机视觉的无人驾驶平台实验仿真系统的实践一共有两个任务，第一个任务是地下停车场任务，该任务作为熟悉任务帮助我们熟悉仿真平台的使用。在此任务中，我们需要选择一个黄色车位作为目标，使小车正确地停在目标车位上。第二个任务是城市街道自动驾驶任务，我们需要在两条既定线路中选择一条作为小车的行驶路线，使小车成功到达线路的终点。

2.1 地下停车场任务

在地下停车场任务中，我们需要依靠小车的左右反光镜、左右窗、前视、后视视角来获取环境中的一些模拟元素，如出入口标志、车道线、地面标志、车位标识等，从而能够判断小车当前所处位置，并选择下一步行动，使得小车最终能够停在任务指定的黄色车位上。

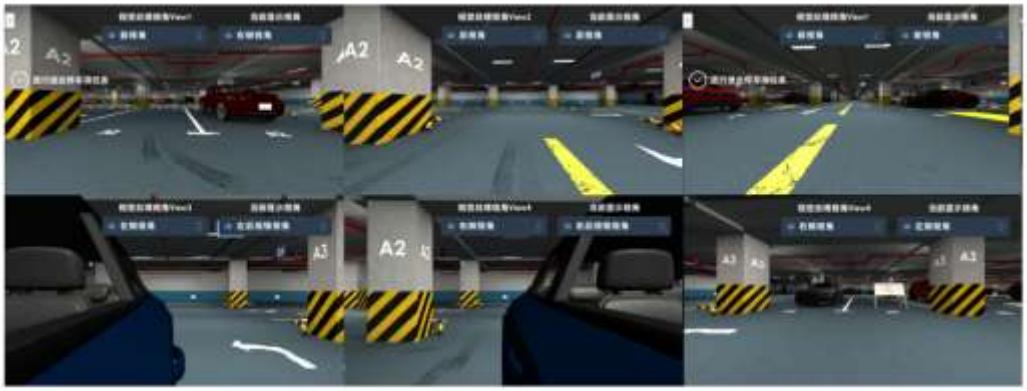

图 6 小车的左右反光镜、左右窗、前视、后视视角

如图 8 所示，停车任务共包括 4 个停车位，均以黄色的“专用车位”地面标识指明，图 1.1.1 中从上到下分别为 A403 侧方停车位、A304 倒车停车位、A203 倒车停车位、A108 斜方停车位。实际可以选择任意一个停车位进行停车任务，4 个停车位并不作难度定位上的区分。

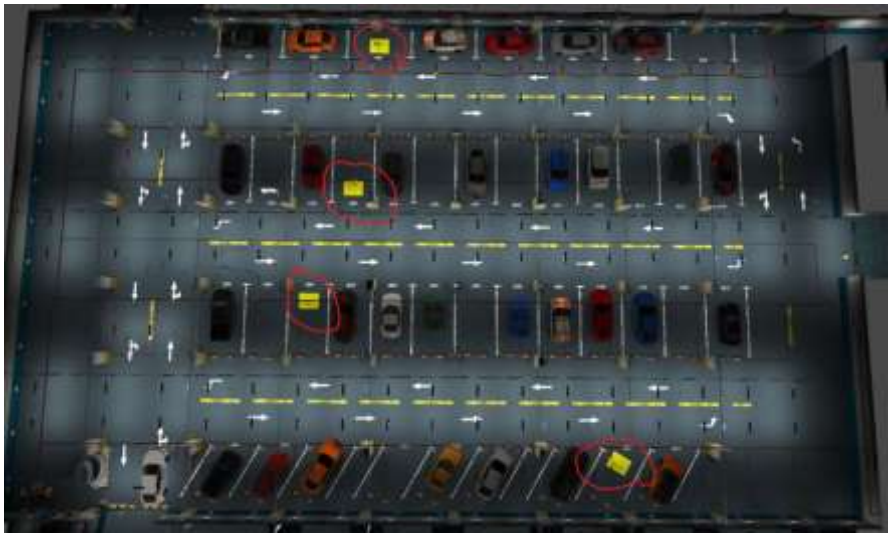

图 8 停车位俯瞰图

车辆从停车场俯瞰图的左下角出发，选择 A203 作为停车位。

车辆出发后，会先经过一个路口，然后在第二个路口时右转。因此第一个问题是要解决路口的判别。由于在每个路口处都会有蓝色的标志牌指示，因此很显然地会将在某个区域内是否检测到蓝色标志牌作为是否到路口的标志。

在转弯过后，就需要直行直至泊车位，泊车的整个流程为先向前行进一段距离，然后

向右，倒车进入泊车位。因此泊车的思路是转弯完成后先直行，直到后视角中某个区域内检测到黄色的标志位，此时停车并进入倒车模式，中途根据时间改变方向，直至完成泊车。

由于实验本身难度不大，因此在每个阶段中使用的都是开环控制。

实验代码如下所示：

（1）颜色定义

首先代码定义了各个颜色的最高和最低阈值，方便后续的颜色识别。

```
color_dist = {'red': {'Lower': np.array([0, 60, 60]), 'Upper': np.array([6, 255, 255])},
              'blue': {'Lower': np.array([80, 100, 120]), 'Upper': np.array([130, 220, 180])},
              'green': {'Lower': np.array([35, 43, 35]), 'Upper': np.array([90, 255, 255])},
              'yellow': {'Lower': np.array([25, 160, 230]), 'Upper': np.array([45, 250, 255])},
              'yellow2': {'Lower': np.array([25, 100, 230]), 'Upper': np.array([45, 180, 255])}}
```

（2）车位识别

根据定义的黄色的阈值将处理好的图片二值化，白色表示处于黄色范围内的像素，而黑色表示其他。最后统计黄色像素点的数量和，如果超过 thres 值，则认为是识别到了车位，输出 1。

```
def Stall_detection(view3):
    # 车位识别
    thres = 1000
    ori1 = view3
    size1 = ori1.shape
    roi1 = ori1[int(150 / 575 * size1[0]):int(500 / 575 * size1[0]), int(800 / 1023 * size1[1]):]
    blur1 = cv2.GaussianBlur(roi1, (5, 5), 0)
    hsv_img1 = cv2.cvtColor(blur1, cv2.COLOR_BGR2HSV)
    kernel = np.ones((3, 3), dtype=np.uint8)
    erode_hsv1 = cv2.erode(hsv_img1, kernel, iterations=1)
    inRange_hsv1 = cv2.inRange(erode_hsv1, color_dist['yellow']['Lower'], color_dist['yellow']['Upper'])
    sum_of_yellow = len((inRange_hsv1[inRange_hsv1 == 255]))
    #cv2.imshow("roi1", roi1)
    #cv2.imshow("hsv1", inRange_hsv1)
    #cv2.waitKey(500)
    print('sum_of_yellow',sum_of_yellow)
    #if sum_of_yellow > 0: print('sum_of_yellow',sum_of_yellow)
    if sum_of_yellow > thres:
        return 1
```

```
    return 0
```

（3）路口识别

首先类似车位识别函数对处理视角图片进行预处理，不同之处在于最后根据蓝色的阈值将处理好的图片二值化，统计蓝色像素点的数量和，如果超过 thres1 值，则认为是识别到了路口，输出 1，代码与车位识别类似。

（4）主函数

主函数根据输入的处理视角的图片，确定车辆每次行动的方向与速度。程序中有三个变量 timer1、counter1，在每次车辆移动后、函数重复运行时都要取上一次运行后的值继续使用，但变量值并不存储在程序中。因此将它们存在外部的文本文档中，每次程序执行开始和结束进行读取和写入。其中 timer1 是一个计时器，作用是对转弯进行开环控制，车辆在转弯时 timer1 值不为 0，且每运行一次递减 1，当 timer1 为 0 时转弯结束，进入下一状态；counter1 代表当前状态，自身数值会随着状态的改变而改变。程序第一次执行时将变量都清零。

```
def image_to_speed(view1, view2, view3, view4, state):
    # global timer1,counter1

    view1 = cv2.imdecode(view1, cv2.IMREAD_ANYCOLOR)
    view2 = cv2.imdecode(view2, cv2.IMREAD_ANYCOLOR)
    view3 = cv2.imdecode(view3, cv2.IMREAD_ANYCOLOR)
    if state.get() is None:
        state.set(0)
    else:
        # print('case: '+str(state.get()))
        state.set(state.get() + 1)
        if state.get() == 1:
            info_tim = open('timer1.txt', 'w')
            info_cnt = open('counter1.txt', 'w')
            info_stp = open('stop1.txt', 'w')
            info_tim.write('0')
            info_cnt.write('0')
            info_stp.write('0')
            info_tim.close()
            info_cnt.close()
            info_stp.close()

    info_tim = open('timer1.txt', 'r')
    info_cnt = open('counter1.txt', 'r')
    info_stp = open('stop1.txt', 'r')
    timer1 = info_tim.read()
    counter1 = info_cnt.read()
    stop1 = info_stp.read()
    info_tim.close()
    info_cnt.close()
```

```
    info_stp.close()
    timer1 = int(timer1)
    counter1 = int(counter1)
    stop1 = int(stop1)
    # print('*', timer1, '*')
    print('counter1:*', counter1, '*')
```

counter1 初始值为 0，检测到路口时值加 1。在到达指定泊车位所在道路并转弯之前需要保持直行，对前视角使用路口检测函数，直到前视角检测到两次路口。因此 counter1 等于 2 时，进行转弯动作，并将 timer1 设为 8，即转弯动作所需要的调试步数为 8 步。timer1 大于 0 期间，左右轮的速度不同，即转弯。转弯动作完成后 counter1 设为 4，保持直行，对后视角使用车位检测函数，直到后视角检测到车位，将 counter1 设为 10。

```
if counter1 <= 2:
        if Crossing_detection(view1)==1:
            print('find crossing!')
            counter1 = counter1+1
            print('counter1:*', counter1, '*')
            if counter1 == 2:
                timer1 = 8
                counter1 = 3
        if counter1 == 0:
            counter1 = 1
    if counter1 ==4:
        if Stall_detection(view2) == 1:
            print('**Stall!**')
            counter1 = 10

    if timer1 == 0:
        left_speed = 1
        right_speed = 1
    else:
        print('timer1',timer1)
        left_speed = 1.323
        right_speed = 1
        timer1=timer1-1
        if timer1 == 0:
            counter1 = 4
```

counter1 大于 10 后，执行倒车操作，之后每次调试步执行后 counter1 加 1，首先倒车至合适位置，然后通过 counter1 的数值开环地调整方向，继续倒车，直至车辆基本平行地泊至车位中，完成泊车操作。由于没有要求开出车库，因此没有进行后续编程。

```
if counter1 >= 10 :
        if counter1 <=14:
            print('**Back!**', counter1)
```

```
            left_speed=-1
            right_speed=-1
            counter1+=1
        elif counter1 <= 19:
            print('**Turning!**', counter1)
            left_speed=-1.4
            right_speed=-0.9
            counter1+=1
        elif counter1 <= 20:
            print('**Turning!**', counter1)
            left_speed=-1.1
            right_speed=-1
            counter1+=1
        elif counter1 <= 23:
            print('**Back!**', counter1)
            left_speed=-1
            right_speed=-1
            counter1+=1
        elif counter1 <= 70:
            print('**GO!**', counter1)
            left_speed=1.1
            right_speed=1
            counter1+=1
        else:
            left_speed=right_speed=0

    if state.get() != 1:
        info_tim = open('timer1.txt','w')
        info_cnt = open('counter1.txt','w')
        info_stp = open('stop1.txt','w')
        info_tim.write(str(timer1))
        info_cnt.write(str(counter1))
        info_stp.write(str(stop1))
        info_tim.close()
        info_cnt.close()
        info_stp.close()

    # constant speed
return left_speed, right_speed
```

实验结果如下所示：

图 9 实验结果

2.2 城市街道自动驾驶任务

在城市街道自动驾驶任务中，我们需要在给定的两条既定路线中选择一条作为小车的行驶线路，在这里，我选择了难度较低的路线 1，该条路线只包含路面标志、斑马线、红绿灯、街道场景等静态元素，这使得我们不需要对一些突发情况(如行人过马路)做特殊处理，只需要按照既定程序行驶即可。经过观察，我们可以发现每个路口均会设置斑马线以标识，因此可以将任务简化为对斑马线的识别，并加入计数器记录斑马线的数目，从而做出相应的动作。但整个过程仍然是开环控制，并没有巡线的操作。

图 10 两条路线的示意图以及难度说明

（1）路口识别

斑马线本身具有很多的特征，比如其颜色特征与等宽度间隔的纹理特征等。在本实验中，由于对识别的要求并不高，因而可以直接提取待处理视角图片的颜色特征来实现对斑马线的

识别。

首先对图片进行裁剪，如果视角里出现斑马线，白色像素的数量会大量增加，只需要设置合适的阈值，就可以进行判断。

```
def Crossing_detection(view1):
    # 路口识别,斑马线
    flag = 0
    thres1 = 10000
    ori = view1
    size = ori.shape
    roi = ori[int(300 / 539 * size[0]):int(400 / 539 * size[0]),
          int(400 / 959 * size[1]):int(900 / 959 * size[1])]
    sum_of_white = len((roi[roi == 255]))
    if sum_of_white != 0:
        print('sum_of_white', sum_of_white)
    if sum_of_white >= thres1:
        flag = 1
    return flag
```

（2）建筑物识别

由于标志点 2 处的位置有一个很明显的红色建筑物，因此只需要设置一定的红色阈值，若超过该阈值则可以开始转弯，具体代码如下所示：

```
def building_detection(view):
    size = view.shape
    roi = view[int(50 / 575 * size[0]):int(250 / 575 * size[0]),
          int(200 / 1023 * size[1]):int(800 / 1023 * size[1])]

    blur2 = cv2.GaussianBlur(roi, (5, 5), 0)
    hsv_img2 = cv2.cvtColor(blur2, cv2.COLOR_BGR2HSV)
    kernel = np.ones((3, 3), dtype=np.uint8)
    erode_hsv2 = cv2.erode(hsv_img2, kernel, iterations=1)
    kernel = np.ones((3, 3), dtype=np.uint8)
    dilate_hsv2 = cv2.dilate(erode_hsv2, kernel, iterations=2)
    inRange_hsv2 = cv2.inRange(dilate_hsv2, color_dist['red']['Lower'
], color_dist['red']['Upper'])
    sum_of_red = len((inRange_hsv2[inRange_hsv2 == 255]))
    print(sum_of_red)
    if sum_of_red >13700:
        return True
```

（3）主函数

主函数实现了对于输入的各个视角的图像，输出对应的控制量：左轮与右轮的速度。

在本实验中使用了同时传入函数的变量 state 来表示当前车辆行驶到的阶段，针对不同的阶段，车辆会做出左转、右转、直行等动作。

首先，state 初值为 0，此时车辆直行，直到车辆的后视角检测到第一个斑马线后左转，之后将继续直行。为了防止循环时对同一条斑马线重复检测，因此每次检测到斑马线都要等待几个循环在进行下一次检测。当检测到建筑物成功后，则左转完成任务。

```
def image_to_speed(view1, view2, view3, view4, state):
    '''''
    view1, view2, view3 format: np.array
    '''
    # format transformation if needed
    view1 = cv2.imdecode(view1, cv2.IMREAD_ANYCOLOR)
    view2 = cv2.imdecode(view2, cv2.IMREAD_ANYCOLOR)

    if state.get() is None:
        state.set(0)
    else:
        # print('case: '+str(state.get()))
        state.set(state.get()+1)
        if state.get() == 1:
            info_tim = open('timer1.txt','w')
            info_cnt = open('counter1.txt','w')
            info_stp = open('stop1.txt','w')
            info_tim.write('0')
            info_cnt.write('0')
            info_stp.write('0')
            info_tim.close()
            info_cnt.close()
            info_stp.close()

    info_tim = open('timer1.txt','r')
    info_cnt = open('counter1.txt','r')
    info_stp = open('stop1.txt','r')
    timer1 = info_tim.read()
    counter1 = info_cnt.read()
    stop1 = info_stp.read()
    info_tim.close()
    info_cnt.close()
    info_stp.close()

    timer1 = int(timer1)
    counter1 = int(counter1)
    stop1 = int(stop1)

    print('*', counter1, '*')

    print('****************************************')
```

```
    print("current cross_counter:{}".format(timer1))
    print('**************************************')
    left_speed = right_speed = 0

    if counter1 <= 1:
        if Crossing_detection(view1)==1:
            print('find crossing!')
            counter1 = counter1+1
            print('*', counter1, '*')
            if counter1 == 1:
                timer1 = 6
                counter1 = 2

    if timer1 == 0:
        left_speed = 5
        right_speed = 5

    if counter1 == 2:
        print('timer1',timer1)
        left_speed = 5
        right_speed = 5
        timer1=timer1-1
        if timer1 == 0:
            counter1 = 3
            timer1 = 5

    if counter1 == 3:
        print('timer1',timer1)
        left_speed = 1.64
        right_speed = 1
        timer1=timer1-1
        if timer1 == 0:
            counter1 = 4

    if counter1 == 4:
        left_speed = 10
        right_speed = 10
        if building_detection(view1) == 1:
            counter1 = 5
            timer1 = 6

    if counter1 == 5:
        print('timer1',timer1)
        left_speed = 5
```

```
86.          right_speed = 5
87.          timer1=timer1-1
88.          if timer1 == 0:
89.              counter1 = 6
90.              timer1 = 9
91.
92.      if counter1 == 6:
93.          print('timer1',timer1)
94.          left_speed = 1
95.          right_speed = 1.3
96.          timer1=timer1-1
97.          if timer1 == 0:
98.              counter1 = 7
99.              timer1 = 15
100.
101.         if counter1 == 7:
102.             print('timer1',timer1)
103.             left_speed = 5
104.             right_speed = 5
105.             timer1=timer1-1
106.             if timer1 == 0:
107.                 counter1 = 8
108.
109.         if counter1 == 8:
110.             print('stop!')
111.             left_speed = 0
112.             right_speed = 0
113.
114.         if state.get() != 1:
115.             info_tim = open('timer1.txt','w')
116.             info_cnt = open('counter1.txt','w')
117.             info_stp = open('stop1.txt','w')
118.             info_tim.write(str(timer1))
119.             info_cnt.write(str(counter1))
120.             info_stp.write(str(stop1))
121.             info_tim.close()
122.             info_cnt.close()
123.             info_stp.close()
124.
125.         # constant speed
126.         return left_speed, right_speed, 0, 0
```

实验结果如下所示：

图 11 实验结果