《 智能系统 》实验报告

学号	姓名	承担任务	贡献度	得分
20215279	李宽宇	共同完成环境配置、实验设计、代码实现、文档	25	
20215027	禹天尹	共同完成环境配置、实验设计、代码实现、文档	25	
20215481	张志维	共同完成环境配置、实验设计、代码实现、文档	25	
20214261	朱齐高	共同完成环境配置、实验设计、代码实现、文档	25	
			共 100	

实验题目	十字路口红绿灯智能控制完整系统实现与测试		
实验时间		**实验地点**	
实验成绩		**实验性质**	☐验证性 ☐设计性 ☑综合性

教师评价：

☐算法/实验过程正确　　☐源程序/实验内容提交

☐程序结构/实验步骤合理　　☐实验结果正确

☐语法、语义正确　　☐报告规范

其他：

评价教师签名：

一、实验目的

为实现十字路口红绿灯智能控制，本次实验的目的是：

（1） 了解上位机与下位机实时通讯原理。

（2） 设计与实现上位机与下位机的实时通讯。

（3） 实现十字路口红绿灯智能控制的完整功能。

报告创建时间：

二、实验项目内容

1、定义上位机与下位机的通讯协议。

2、设计实现上位机的串口通讯程序。

3、设计实现下位机的串口通讯程序。

4、包含上下位机的完整系统的实现与测试。

三、实验过程或算法（代码）

1、定义上位机与下位机的通讯协议 本次实验共使用1个红外避障模块与2个交通灯模块，它们的引脚连 接表如下：

交通灯模块（南北交通灯）：

交通灯模块引脚	Arduino UNO R3引脚
GND	GND
R	12号数字引脚
Y	不接入
G	13号数字引脚

交通灯模块（东西交通灯）：

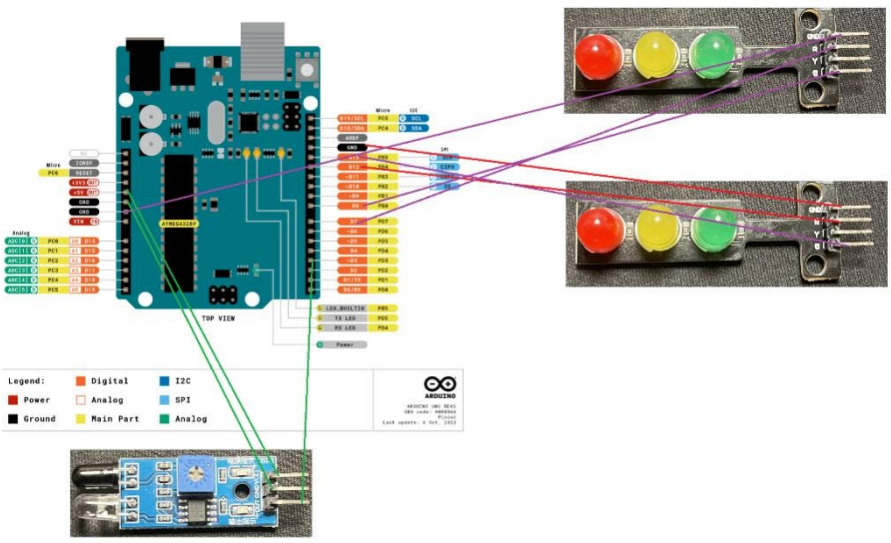
交通灯模块引脚	Arduino UNO R3引脚
GND	GND
R	9号数字引脚
Y	不接入
G	10号数字引脚

红外避障模块：

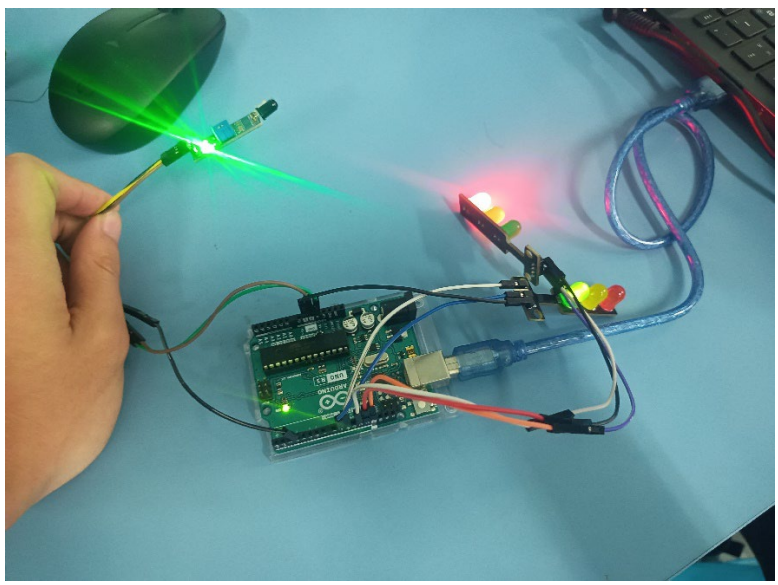
红外模块引脚	Arduino UNO R3引脚

VCC	5V供电引脚
GND	GND
OUT	3号数字引脚

结合官方引脚定义说明：



实物连接图：



需求分析：

在本实验中，需要使用下位机采集传感器信息，传输给上位机，上位机需要利用采集得到的信息进行推理，并将推理结果传给下位机，由下位机再传给交通灯模块，实现十字路口红绿灯智能控制。

下位机传给上位机的信息：

只有红外避障模块的检测信息，通 过0、1（ASCII编码）串的形式传输给上位机。

上位机传给下位机的信息：

十字路口的红绿灯组包含南北方向 的红绿灯和东西方向的红绿灯，这两个方向的红绿灯开关是互斥的， 故红绿灯的开关情况只有2种，1种是南北红绿灯是绿灯，东西红绿 灯是红灯，还有1种是南北红绿灯是红灯，东西红绿灯是绿灯。故也 可只用0、1（ASCII编码）串的形式通知下位机当前时刻红绿灯组的 状态——0代表南北红绿灯是红灯、东西红绿灯是绿灯，1代表南北 红绿灯是绿灯、东西红绿灯是红灯。

2、设计实现上位机的串口通讯程序

上位机读取下位机的红外避障模块检测结果：

```python
# 读串口线程
def read_serial(event):
    while not event.is_set():
        try:
            # 读取串口数据
            data = ser.readline().decode('utf-8').rstrip()
            # 将读取的数据写入显示缓冲区
            if data:
                print(data)
                if data == "NSYES":
                    WECARS[0] = WECARS[0] + 1
                if data == "WEYES":
                    NSCARS[0] = NSCARS[0] + 1
        except Exception as e:
            pass
    print("Read Serial Exit!")
```

上位机读取串口数据，并根据读到的字符串来进行南北车流量的计数。如果读取到的字符串是NSYES，则南北方向车流量增加一，如果读取到的字符串是WEYES，则东西方向车流量加一。

上位机控制下位机的红绿灯组信号：

```python
    """
    计时，每秒都会输出时间
    """
    @Rule(AS.oldFact << Fact(Ticks=MATCH.times),
          Fact(NSsign=MATCH.NScolor),
    )
    def ticks(self, times, oldFact,NScolor):
        time.sleep(1)
        self.retract(oldFact)
        self.declare(Fact(Ticks=times + 1))
        print("====== 启动时间{}秒{} ======".format(times,NScolor))
        if NScolor == 'GREEN':
            output_queue.put(b"1")
        else:
            output_queue.put(b"0")
```

```python
# 写串口线程
def write_serial(event):
    while not event.is_set():
        try:
            # 从输出队列中读取数据
            # [Note] 如果队列为空，会阻塞，直到有数据写入队列。因此要设置一个超时时间
            data = output_queue.get(timeout=0.5)
            # 将数据写入串口
            if data:
                ser.write(data)
        except Exception as e:
            pass
    print("Write Serial Exit!")
```

每秒钟，上位机都会向输出队列里面加入0或1，写串口进程一旦检测到有数据写入队列，就将其写入串口。

3、**设计实现下位机的串口通讯程序**

接口定义：

```
int Led = 13;        //定义绿灯LED 接口
int buttonpin = 3;   //定义避障传感器接口
int val;             //定义避障传感器接口的数字变量val
int prv = -1;        //定义前一次传感器收到的val
char chr;            //定义上位机的输入字符信号
int LedRED = 12;       //定义红灯LED 接口
int Led2 = 10;
int Led2RED = 9;
int tmp_chr = 1;
void setup() {
  pinMode(Led, OUTPUT);        //定义LED 为输出接口
  pinMode(LedRED, OUTPUT);       //定义LED 为输出接口
  pinMode(Led2, OUTPUT);       //定义LED 为输出接口
  pinMode(Led2RED, OUTPUT);       //定义LED 为输出接口
  pinMode(buttonpin, INPUT);   //定义避障传感器为输出接口
  Serial.begin(9600);          //连接上位机，波特率为9600
  digitalWrite(Led, LOW);
  digitalWrite(LedRED, HIGH);
  digitalWrite(Led2, HIGH);
  digitalWrite(Led2RED, LOW);
}
```

在下位机代码的开头定义了两个交通灯模块和红外避障模块的 引脚常量，并在 setup 函数中定义了它们的输入输出属性。

串口通讯与解析：

```
void loop() {
  val = digitalRead(buttonpin);  //将数字接口3的值读取赋给val
  if (val != prv) { // 只有当val发生变化的时候才向上位机发数据
    if (val == LOW) {  //当避障传感器检测低电平时，有障碍物，传输YES
      if (tmp_chr == '1') {
        Serial.print("NSYES\n");
      }
      else if (tmp_chr == '0') {
        Serial.print("WEYES\n");
      }
    }
    else {
      Serial.print("NO\n");
    }
  }
  prv = val;
  // 接收上位机信号控制小灯开关
  chr = Serial.read();
  if (chr == '1') {  //当避障传感器检测低电平时，LED 灭
    digitalWrite(Led, HIGH);
    digitalWrite(LedRED, LOW);
    digitalWrite(Led2, LOW);
    digitalWrite(Led2RED, HIGH);
    tmp_chr = chr;
  }  else if (chr == '0') {
    digitalWrite(Led, LOW);
    digitalWrite(LedRED, HIGH);
    digitalWrite(Led2, HIGH);
    digitalWrite(Led2RED, LOW);
    tmp_chr = chr;
  }
}
```

在每个循环中先读取红外避障模块的检测结果 button_pin，如果当前检测结果与上次循环的检测结果不同，并且当前检测结果是低电平，则说明传感器检测到了障碍物，随后根据上次循环中从上位机读取的红绿灯控制指令，来决定上传“NSYES”还是“WEYES”。然后读取上位机发送来的红绿灯组控制指令，若为 1，则代表南北红绿灯为绿灯，东西红绿灯为红灯。若为 0，则代表南北红绿灯为 红灯，东西红绿灯为绿灯。按照指令将对应的输出引脚置为对应的高低电平。

4、包含上下位机的完整系统的实现与测试

本章节主要描述为实现完整系统，而对实验2中知识引擎所进行 的修改。

知识引擎初始化的修改：

```
# 创建三个缓冲区
buffer_display = Buffer()   # 用于显示上方提示信息
buffer_output = Buffer()    # 用于显示从串口读入的数据
buffer_input = Buffer()     # 用于从用户读入要向串口写入的数据

# 串口参数设置
serialPort = "COM3" # 串口
baudRate = 9600     # 波特率
buffer_display.text = "输入`exit`退出程序，输入`1`或`0`控制交通灯亮灭\n"
buffer_display.text += "参数设置：串口=%s ，波特率=%d" % (serialPort, baudRate)

# 连接串口
try:
    ser = serial.Serial(serialPort, baudRate, timeout=0.5)
except:
    raise Exception("无法打开串口！请确保串口号正确，且设备已连接！")

# 创建输出队列，存储要向串口写入的数据
output_queue = Queue()

# 创建终止标志变量，用于Control-C退出时终止线程
stop_event = threading.Event()
```

```
if __name__ == '__main__':
    # 启动线程
    t1 = threading.Thread(target=read_serial, args=(stop_event,))
    t2 = threading.Thread(target=write_serial, args=(stop_event,))
```

在知识引擎的初始化函数中，添加了串口连接的建立，后台串口读取和写入线程的启动。

车辆计数逻辑的修改：

```
            # 读取串口数据
            data = ser.readline().decode('utf-8').rstrip()
            # 将读取的数据写入显示缓冲区
            if data:
                print(data)
                if data == "NSYES":
                    WECARS[0] = WECARS[0] + 1
                if data == "WEYES":
                    NSCARS[0] = NSCARS[0] + 1
```

修改原本使用随机数增加南北、东西车辆的逻辑，替换为读取串口数据来增加南北、东西通过车辆数。

计时器函数的修改：

```
    """
    计时，每秒都会输出时间
    """
    @Rule(AS.oldFact << Fact(Ticks=MATCH.times),
          Fact(NSsign=MATCH.NScolor),
    )
    def ticks(self, times, oldFact,NScolor):
        time.sleep(1)
        self.retract(oldFact)
        self.declare(Fact(Ticks=times + 1))
        print("====== 启动时间{}秒{} ======".format(times,NScolor))
        if NScolor == 'GREEN':
            output_queue.put(b"1")
        else:
            output_queue.put(b"0")
```

在原本每秒输出时间的计时器函数中插入对串口写入数据的更新逻辑，每秒中都会向输出队列写入数据。

时间分配逻辑的修改：

```
    @Rule(
        Fact(Ticks=MATCH.times),
        AS.oldFact1 << Fact(switchTime=MATCH.switchTime),
        AS.oldFact2 << Fact(NeedChange=True),
        salience=2
    )
    def changeSwitchTime(self, oldFact1, oldFact2, times, switchTime):
        # print("修改时间")
        self.retract(oldFact1)
        self.retract(oldFact2)
        temp = []
        temp.append(NSCARS[0])
        temp.append(WECARS[0])
        CARS.append(temp)
        NSCARS[0] = 0
        WECARS[0] = 0
        cur_epoch = times // PERIOD
        newSwitchTime = switchTime
        # 修正红绿灯
        print("第 {} 个周期，南北方向通过{}辆车，东西方向通过{}辆车".format(cur_epoch, CARS[cur_epoch][1], CARS[cur_epoch][0]))
        if CARS[cur_epoch][0]==0 and CARS[cur_epoch][1]==0:
            newSwitchTime = 5
        else:
            newSwitchTime = math.floor(PERIOD / (CARS[cur_epoch][0] / (CARS[cur_epoch][1] + 0.00001) + 1))
        if newSwitchTime < 3:
            newSwitchTime = 3
        if newSwitchTime > 7:
            newSwitchTime =7;
        self.declare(Fact(switchTime=newSwitchTime))
        self.declare(Fact(switch=True))
        self.declare(Fact(NeedShow=True))
```

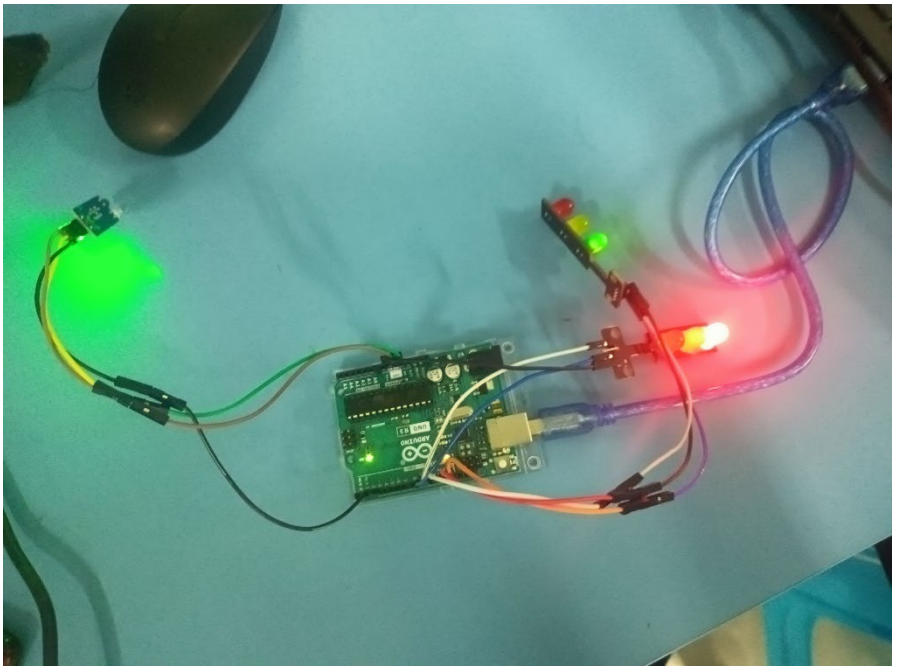
修改两个方向红绿灯的时间分配逻辑，不再是从随机生成的CARS列

表中读取每个周期两个方向的车流量，而是根据当前周期两个方向的车流量进行时间分配，并且时间分配的比例有上下界，即每个方向的时间都必须介于3-7s之间。

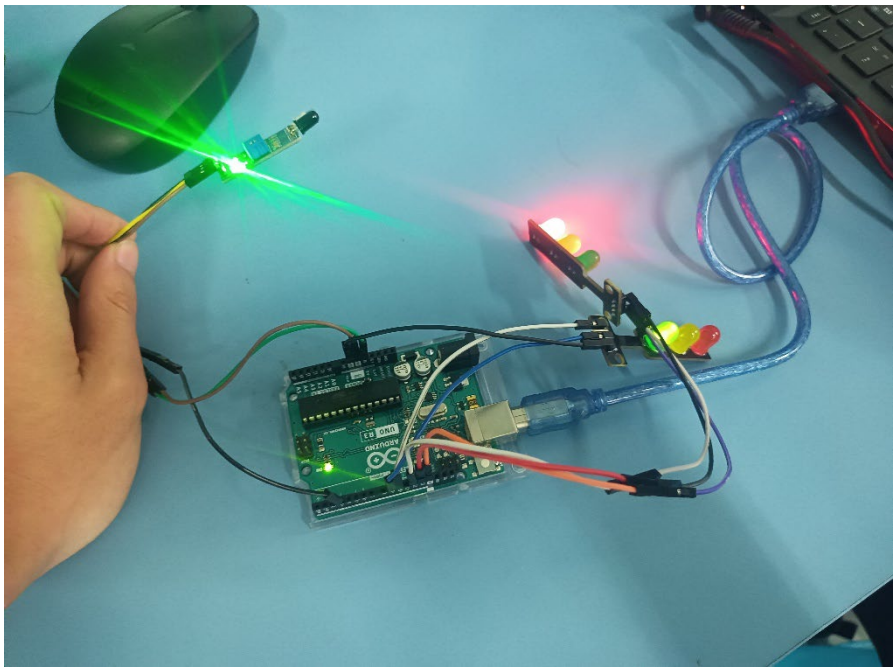
四、实验结果及分析

红绿灯组的两种状态：

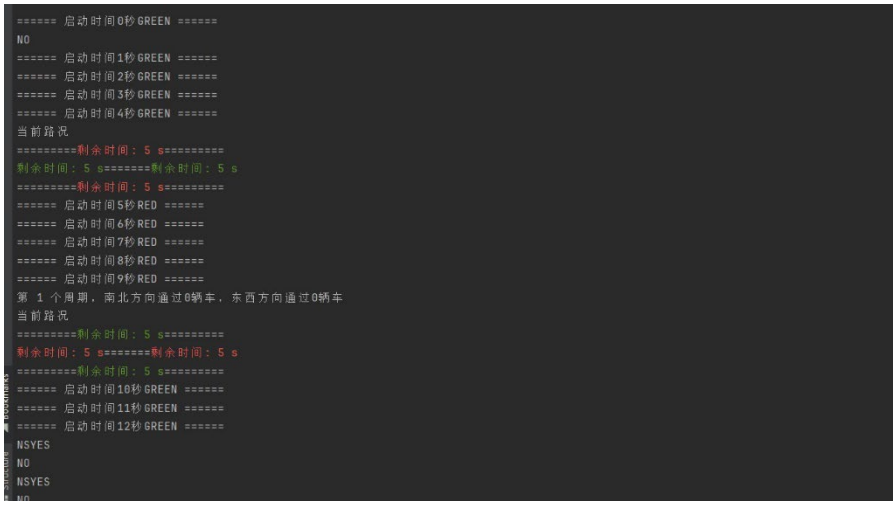
南北为红灯，东西为绿灯：



南北为绿灯，东西为红灯：

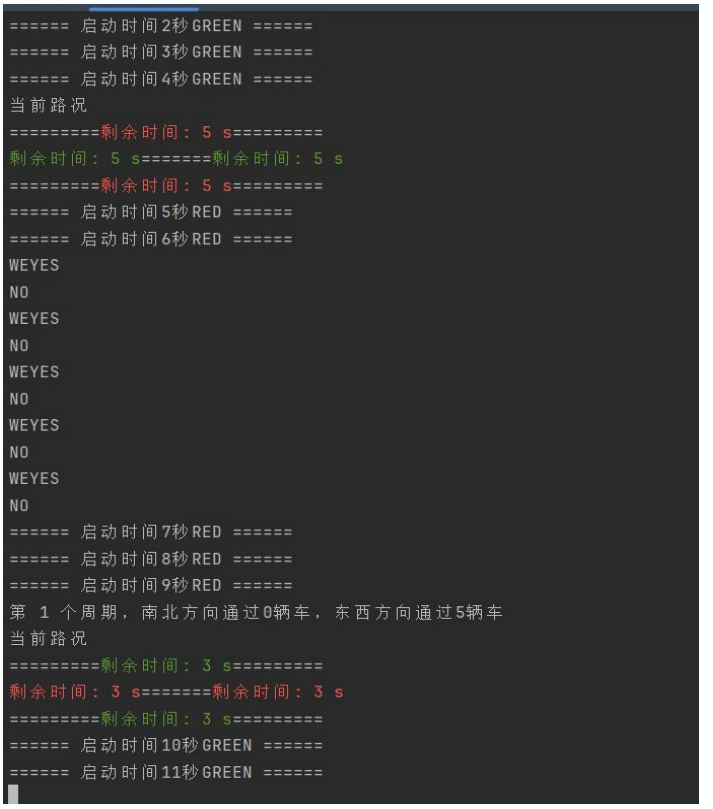


南北路口与东西路口都没有车辆通过：



由于当前周期两个方向都没有车辆经过，故两个方向下个周期的绿灯时间均为 5s

东西方向车流量大于南北方向：



由于当前周期东西方向车流量大于南北方向，故下个周期南北方向绿灯时间要小于东西方向的绿灯时间。

评价总结：

该十字路口红绿灯智能控制系统在实验环境中表现优秀，能够根据南北和东西路口的车流量动态调整红绿灯时长。它为解决实际交通场景中因静态红绿灯时间导致的交通拥堵问题提供了有效的解决方案。 系统的核心在于，它可以通过分析一轮红绿灯周期中的车辆数来确定下一轮中两个方向红绿灯的时长。通过车辆数多的方向将享有更长的绿灯时间，而车辆数少的方向则会有更长的红灯时间。这种动态调整机制能够更好地优化交通流量，避免交通堵塞。

特色功能和创新点：

（1）该系统给两个方向的红绿灯时间设定了上下限，两个方向的红

绿灯时间始终位于 3-7s 之间。

（2）此外，该系统还在检测车辆是否通过路口时实现了“防抖”机制， 在红外避障检测模块被遮挡时，并不会在短时间内发送大量的信号。

缺点和不足：

（1）然而，系统在实验和实际场景之间存在一定的差距。例如，实验中设置的每轮红绿灯时间为 10 秒，而在实际情况中这种情况较少。 这使得系统的红绿灯时间划分粒度较粗，可能无法细致地反映出两个路口车辆数的差异。

（2）由于上位机和下位机的系统时钟存在误差，随着系统持续运行时间的增长，这种误差会使得上下位机的信号传递产生发生延迟甚至是错误。

总的来说，这是一个具有潜力的系统，能够提供有效的交通管 理解决方案。然而，为了使其更好地适应实际情况，还需要进一步改进和优化，例如改进红绿灯时间划分的粒度，使上下位机的系统时钟保持一致，来更准确地检测车辆流量。

五、完成时间

（1）实验时间：2021.6.5，2021.6.19

（2）检查时间：2021.6.14

（3）2021 年 6 月 19 日 23:59 之前提交实验报告