**《 智能系统 》实验报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学号** | | **姓名** | **承担任务** | | | **贡献度** | **得分** |
| **20194053** | | **王志鹏** | **共同完成系统集成** | | |  |  |
| **20194051** | | **王良厅** | **共同完成系统集成，实验报告4的撰写** | | |  |  |
| **20184199** | | **王赛宇** | **共同完成系统集成，新添加了解释器管理UI** | | |  |  |
|  | |  |  | | |  |  |
|  | |  |  | | |  |  |
|  | |  |  | | |  |  |
|  | |  |  | | |  |  |
| **实验题目** | **系统集成** | | | | | | |
| **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： | | | | | | | |
| 1. 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）完成项目集成  （2）实现合理的界面 | | | | | | | |
| 1. 实验项目内容   1、项目集成  （1）传感器数据采集；  （2）下位机数据封装上传；  （3）上位机数据管理；  （4）知识库管理与维护；  （5）推理机与解释器设计和实现；  （6）数据封装下发；  （7）传感器响应；  （8）测试“传感器采集——传输——数据库——知识库——推理机——数据库——传输——传感器响应”的全过程。  2、系统可视化  （1）设计系统UI；  （2）实现系统UI；  （3）通过UI实现对系统的管理维护和运行仿真。  3、实时交通路口仿真（附加）  （1）获取地图软件（高德、百度等）中的路况信息；  （2）解析路况信息；  （3）通过UI实现交通路口红绿灯仿真。  \* 系统UI参考《程序基本界面.exe》 | | | | | | | |
| 三、实验过程或算法（代码）  1、项目集成  将四个红绿灯以及避障传感器分别接在下位机上，下位机通过连接线连接上位机（PC）的USB接口。    传感器连接情况   |  |  |  | | --- | --- | --- | | 传感器 | 端口 | 说明 | | 避障传感器 | OUT连接在下位机的10接口 |  | | 红绿灯1 | green连2，red连3 | 显示南北右转情况 | | 红绿灯2 | green连6，red连7 | 东西左转 | | 红绿灯3 | green连9，red连8 | 东西右转 | | 红绿灯4 | green连12，red连13 | 南北左转 |   下位机数据采集  （1）设计传感器数据采集方案；  避障传感器：检测到有障碍物向上位机发送’yes’，没有则发送’no’  （2）设计并实现数据采集程序（函数）。  int last = 1;  int now = 1;  void loop() {  //判断有无障碍 1：有障碍 0： 无障碍  int now = digitalRead(10);  if(now != last){  if(now == 0){  Serial.print("yes");  //digitalWrite(LED\_BUILTIN, HIGH);  }  else{  Serial.print("no");  //digitalWrite(LED\_BUILTIN, LOW);  }  }  last = now;  （3）数据编码与传输  在下位机采集到传感器数据后，通过Serial.print()函数将数据传输给上位机 。  在上位机中serial中的read函数读取来自下位机的数据。  def checkTask():  last = ""  while(1):  data = ser.readline()  s3 = data.decode().rstrip()  print(s3)  if s3 == 'yes':  for client in SocketHandler.clients:  client.write\_message(json.dumps({  'type': "addCar",  "data": True  }))  last = s3  （4）数据解析与输出  （1）设计信号灯显示方案；  下位机接受来自上位机发送的数据，接收到信号灯状态信息，然后再在下位机中向传感器发送控制信息。   |  |  | | --- | --- | | 信号灯状态码 | 说明 | | 0 | 控制南北右转方向的绿灯亮 | | 1 | 控制东西左转方向的绿灯亮 | | 2 | 控制东西右转方向的绿灯亮 | | 3 | 控制南北左转方向的绿灯亮 |   代码：  String ctrl = Serial.readString();  Serial.println(ctrl);  if(ctrl[0]=='0' ){  //南北  Serial.println("ctrl=='0'");  digitalWrite(3, LOW);  digitalWrite(2, HIGH);  digitalWrite(6, LOW);  digitalWrite(7, LOW);  digitalWrite(8, LOW);  digitalWrite(9, LOW);  digitalWrite(12, LOW);  digitalWrite(13, LOW);    }  if(ctrl[0]=='1' ){  Serial.println("ctrl=='1'");  digitalWrite(3, LOW);  digitalWrite(2, LOW);  digitalWrite(6, HIGH);  digitalWrite(7, LOW);  digitalWrite(8, LOW);  digitalWrite(9, LOW);  digitalWrite(12, LOW);  digitalWrite(13, LOW);  }  if(ctrl[0]=='2' ){  Serial.println("ctrl=='2'");  digitalWrite(3, LOW);  digitalWrite(2, LOW);  digitalWrite(6, LOW);  digitalWrite(7, LOW);  digitalWrite(8, LOW);  digitalWrite(9, HIGH);  digitalWrite(12, LOW);  digitalWrite(13, LOW);  }  if(ctrl[0]=='3' ){  Serial.println("ctrl=='3'");  digitalWrite(3, LOW);  digitalWrite(2, LOW);  digitalWrite(6, LOW);  digitalWrite(7, LOW);  digitalWrite(8, LOW);  digitalWrite(9, LOW);  digitalWrite(12, HIGH);  digitalWrite(13, LOW);  }  （5）知识库管理与维护  有可信度和模糊知识；  其中可信度知识结构：   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 编号 | 前提 | 结论 | 前提可信度 | 知识可信度 |   模糊度知识结构：   |  |  |  |  | | --- | --- | --- | --- | | 编号 | 名称 | 输入变量到隶属度矩阵 | 模糊矩阵 |   可信度知识：    添加界面：    模糊知识：    添加：    （6）推理机与解释器  可信度知识推理:  由于在程序当中需要将初始值控制在[0,1]的范围之内，因此我们按照如下定义来处理：  推理过程：   1. **def** confident\_infer(rain, snow, wind, lightTime): 2. **print**("chushi:", lightTime) 3. rain = max(0, min(100, rain)) 4. snow = max(0, min(100, snow)) 5. wind = max(0, min(100, wind)) 6. cf\_p = dict() 7. cf\_p['下雨'] = rain / 100 8. cf\_p['下雪'] = snow / 100 9. cf\_p['大风'] = wind / 100 10. newdict = {} 11. entitys = dbUtil.findKLList(db) 12. **for** index, element **in** enumerate(entitys): 13. premise = element.E 14. conclusion = element.H 15. val = cf\_p[premise] 16. **if** val >= element.CFE: 17. newdict[conclusion] = val \* element.CFHE 18. **print**("所有相关结论：", newdict) 19. maxkey = "" 20. **for** key, value **in** newdict.items(): 21. **if** (value == max(newdict.values())): 22. maxkey = key 23. **if** (maxkey == "绿灯时间+3s"): 24. **print**("最终结论为：", maxkey) 25. **for** i **in** range(len(lightTime)): 26. lightTime[i] += 3 27. **elif** (maxkey == "绿灯时间+5s"): 28. **print**("最终结论为：", maxkey) 29. **for** i **in** range(len(lightTime)): 30. lightTime[i] += 5 31. **elif** (maxkey == "绿灯时间+6s"): 32. **print**("最终结论为：", maxkey) 33. **for** i **in** range(len(lightTime)): 34. lightTime[i] += 6 35. # print("zuihou", lightTime) 36. **return** lightTime   在上述代码中，我们首先初始化了3类前提的阈值，然后将输入的条件与阈值相比较，只有当条件可能性大于阈值，我们才考虑进行可信度的推理。随后我们遍历整个知识库，将每条知识都做相应的可信度推理，并将其结果存入一个字典。结束知识遍历之后，我们从字典当中选择值最大的，也就是可能性最高的结论作为我们可信度推理的最终结论，并将修改后的红绿灯时长传送给前台。  模糊知识推理：  模糊推理的基本方法是：根据车道上的车辆的数量，进行模糊权重估计，根据权重来分配一个总的通勤时常。为了避免通勤时常过短，还加入了一个基础的通勤时常。总体代码如下图所示；   1. *# 计90s为一次通行周期，根据权重分配通行时间.* 2. *# 基础通行时间为5s，在5s基础上累加.* 3. total\_time = 60 4. basePassTime = 5 5. fuzzy\_handler = fuzzy.get\_default\_fuzzy() 6. topRightWeight = fuzzy\_handler.get\_result\_by\_input(get\_fuzzy\_name\_by\_car\_number(topRight)) 7. eastLeftWeight = fuzzy\_handler.get\_result\_by\_input(get\_fuzzy\_name\_by\_car\_number(eastLeft)) 8. eastRightWeight = fuzzy\_handler.get\_result\_by\_input(get\_fuzzy\_name\_by\_car\_number(eastRight)) 9. topLeftWeight = fuzzy\_handler.get\_result\_by\_input(get\_fuzzy\_name\_by\_car\_number(topLeft)) 10. total\_weight = topLeftWeight + topRightWeight + eastLeftWeight + eastRightWeight 11. pass\_times = [ 12. basePassTime + (total\_time \* topRightWeight / total\_weight), 13. basePassTime + (total\_time \* eastLeftWeight / total\_weight), 14. basePassTime + (total\_time \* eastRightWeight / total\_weight), 15. basePassTime + (total\_time \* topLeftWeight / total\_weight), 16. ]   其中推理过程如下方代码所示；   1. def compute\_fuzzy\_result(input\_membership\_vec, matrix\_r): 2. """ 3. compute fuzzy result of fuzzy reasoning. 4. example: 5. input\_membership\_vec: [0.8, 0.2, 0, 0] 6. matrix\_r: [ 7. [0.8, 0.2, 0, 0], 8. [0.2, 0.5, 0.5, 0], 9. [0, 0.5, 0.5, 0.2], 10. [0, 0, 0.2, 0.8] 11. ] 12. return: 0.2 = 0.8 \* 0 + 0.2 \* 1 + 0 \* 2 + 0 \* 3 13. """ 14. res = 0 15. for inx\_vec, val\_vec in enumerate(input\_membership\_vec): 16. min\_result = [min(val\_vec, row[inx\_vec]) for row in matrix\_r] 17. res += inx\_vec \* max(min\_result) 18. return res 19. class Fuzzy: 20. def \_\_init\_\_(self, map\_input\_2\_membership\_vec, matrix\_r): 21. """ 22. init create a fuzzy object from orm.Fuzzy items 23. """ 24. self.\_\_input2vec = json.loads(map\_input\_2\_membership\_vec) 25. self.\_\_r = json.loads(matrix\_r) 26. def get\_result\_by\_input(self, in\_name): 27. """ 28. get fuzzy result by input name. 29. example: 30. :in\_name: "small" 31. :return: 0.2 32. """ 33. if self.\_\_input2vec.get(in\_name) is None: 34. raise "input name not found in map\_input\_2\_membership\_vec" 35. return compute\_fuzzy\_result(self.\_\_input2vec[in\_name], self.\_\_r)   解释器部分：  每一次推理结束，将推理的前提和结论以及使用到的知识存入数据库中。   1. dbUtil.insertResult( 2. db=db, 3. name="", 4. weather\_rain=res\_weather\_rain, 5. weather\_snow=res\_weather\_snow, 6. weather\_wind=res\_weather\_wind, 7. base\_pass\_time=res\_base\_pass\_time, 8. public\_time=res\_public\_time, 9. car\_numbers=res\_car\_numbers, 10. public\_allocated\_times=res\_public\_allocated\_times, 11. weather\_add\_times=res\_weather\_add\_times, 12. final\_results=res\_final\_results, 13. time=res\_time, 14. )   2系统UI:   1. 车流量管理     2.天气管理（用于可信度知识推理）      3）推理结果（用于解释器） | | | | | | | |
| 1. 实验结果及分析   测试：  当rain=20，snow=20，wind=20时候，检测到4个方向上的车流量分别为4，4，8，1时，推理得出红绿灯时间为 [7.37037037037037, 7.37037037037037, 13.888888888888888, 7.37037037037037]。  如下图所示： | | | | | | | |
| 1. 完成时间   （1）实验时间：2022.5.13，2022.5.20  （2）检查时间：2022.5.20  （3）2022年5月24日23:59之前提交实验报告 | | | | | | | |