

**本科毕业设计（论文）**



|  |  |
| --- | --- |
| **题目** | **楼宇火灾无线监测报警** |
| **系统的设计** | |

|  |  |  |  |
| --- | --- | --- | --- |
| **学生姓名** | **任思霖** | **学　号** | **201431030690** |
| **教学院系** | **机电工程学院** | | |
| **专业年级** | **测控技术与仪器2014级** | | |
| **指导教师** | **罗宾** | **职　称** | **高级实验师** |
| **单　　位** | **西南石油大学** | | |
| **辅导教师** |  | **职 称** |  |
| **单　　位** |  | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **完成日期** | **2018** | **年** | **5** | **月** | **6** | **日** |

Southwest Petroleum University

Graduation Thesis



**Design of Building Fire Wireless Monitoring and Alarm System**

**Grade: 2014**

**Name: Ren SiLin**

**Speciality: Measurement Control**

**Technology and Instruments**

**Instructor: Luo Bin**

**School of Mechanical and Electrical Engineering**

**2018-5**

# 摘要

在社会高速发展的今天，建筑的外观与质量已经得到了很大的提升，与此同时，由于建筑的密集度增大与电器设备的规模化运用，建筑物本身所受到的威胁也越来越大，这导致增加了火灾发生的概率。

本设计中提出了一种能够监测温度，一氧化碳（CO）浓度与可燃性气体烟雾浓度的火灾监测系统。该设计结合了无线传感器网络技术，选取ZigBee无线通信技术来完成传输，采用ZigBee的最佳片上解决方案CC2530作为终端（采集数据）与协调器（收发数据）。

硬件系统包含三个部分：集成传感器采集模块、ZigBee传输模块与主控模块；软件系统运行在PC端上，实时读取串口数据，将传感器采集到的数据进行融合。融合算法是基于遗传算法优化的BP神经网络算法与模糊逻辑算法做出的各种决策。经过数据融合算法处理后的决策值有效度大于传统设备，极大程度上避免了传统设备因传感器单方面的数据输入造成精度不高与误报产生的情况出现。

本文在理论的基础上做了相应的结果论证。试验数据表明，通过传感器数据融合算法后的数据精确度比传统设备数据的精度高，且能够对阴火和明火火灾统一报警。本系统确定了研究方向，且系统在实验中工作稳定，数据可靠，满足火灾报警的需求。

**关键词：**ZigBee；火灾探测；数据融合；神经网络算法；

# Abstract

With the development of society, the appearance and quality of buildings have been greatly improved. At the same time, due to the increase in the density of buildings and the large-scale use of electrical equipment, the threat to the buildings themselves will increase. And it shows that the probability of fire is increasing.

In this design, a fire monitoring system capable of monitoring temperature, carbon monoxide (CO) concentration and flammable gas smoke concentration is proposed. The design incorporates wireless sensor network technology, uses ZigBee protocol for wireless data transmission, and uses ZigBee's best on-chip solution CC2530 as a terminal (acquisition data) and coordinator (transceiver data).

The hardware system consists of three parts: integrated sensor acquisition module, ZigBee transceiver module and master module; the software system runs on the PC, reads the serial data in real time, and carries out fusion processing of the sensor data. The fusion algorithm is a BP neural network algorithm optimized based on genetic algorithms. After the fuzzy rules, various decision-making decisions are made. The decision value processed by the data fusion algorithm is more effective than the traditional device, Therefore, the situation that the accuracy of the conventional device is not high and the false alarm is generated which caused by the unilateral data input of the sensor is avoided.

This article makes the corresponding result demonstration on the basis of the theory. According to the experimental results, the accuracy of data after sensor data fusion algorithm is higher than that of traditional equipment data, and can uniformly warn the fires of naked fire and open flame. The system determines the research direction, and the system is stable in the experiment, the data is reliable, and it meets the needs of fire alarms.

**Keywords:** ZigBee; fire detection; data fusion; neural network algorithm;

**目录**

[摘要 I](#_Toc514841083)

[Abstract II](#_Toc514841084)

[1 绪论 1](#_Toc514841085)

[1.1 论文的研究背景及意义 1](#_Toc514841086)

[1.2 国内外研究现状 1](#_Toc514841087)

[1.2.1 国内研究现状 2](#_Toc514841088)

[1.2.2 国外研究现状 3](#_Toc514841089)

[1.3 论文的主要研究内容 3](#_Toc514841090)

[2 无线传感器网络技术及ZigBee简介 5](#_Toc514841091)

[2.1 无线传感器网络技术 5](#_Toc514841092)

[2.1.1 无线传感器网络体系结构 5](#_Toc514841093)

[2.1.2 无线传感器网络的特点 6](#_Toc514841094)

[2.2 ZigBee简介 7](#_Toc514841095)

[2.2.1 ZigBee技术特点 7](#_Toc514841096)

[2.2.2 ZigBee协议栈架构 8](#_Toc514841097)

[2.2.3 ZigBee与其他无线标准的技术特点比较 12](#_Toc514841098)

[2.3 本章小结 12](#_Toc514841099)

[3 火灾报警系统的数据处理方法 13](#_Toc514841100)

[3.1 相关算法介绍 13](#_Toc514841101)

[3.1.1 遗传算法 13](#_Toc514841102)

[3.1.2 神经网络算法 16](#_Toc514841103)

[3.1.3 模糊逻辑算法 21](#_Toc514841104)

[3.2 信息融合技术 21](#_Toc514841105)

[3.3 信息融合技术在火灾报警系统中的应用 22](#_Toc514841106)

[3.3.1 数据层融合器 23](#_Toc514841107)

[3.3.2 特征层融合器 24](#_Toc514841108)

[3.3.3 决策层融合器 27](#_Toc514841109)

[3.4 本章小结 28](#_Toc514841110)

[4 火灾报警系统的总体方案设计 29](#_Toc514841111)

[4.1 系统设计原则 29](#_Toc514841112)

[4.2 选型分析 29](#_Toc514841113)

[4.2.1 处理器选型 29](#_Toc514841114)

[4.2.2 节点传感器选型 30](#_Toc514841115)

[4.3 系统总体方案设计 31](#_Toc514841116)

[4.3.1 系统硬件总体架构 31](#_Toc514841117)

[4.3.2 系统软件总体架构 32](#_Toc514841118)

[4.4 本章小结 33](#_Toc514841119)

[5 火灾报警系统的硬件设计 34](#_Toc514841120)

[5.1 总体硬件设计思路 34](#_Toc514841121)

[5.2 CC2530最小系统硬件电路 34](#_Toc514841122)

[5.3 终端节点硬件电路设计 36](#_Toc514841123)

[5.3.1 温度传感器节点 36](#_Toc514841124)

[5.3.2 烟雾传感器节点 36](#_Toc514841125)

[5.3.3 一氧化碳探测器节点 38](#_Toc514841126)

[5.3.4 AD转换设计 39](#_Toc514841127)

[5.4 协调器节点硬件电路设计 39](#_Toc514841128)

[5.5本章小结 40](#_Toc514841129)

[6 火灾报警系统的软件设计 41](#_Toc514841130)

[6.1 软件开发平台 41](#_Toc514841131)

[6.1.1 IAR嵌入式软件开发平台 41](#_Toc514841132)

[6.1.2 ZStack协议栈开发平台 41](#_Toc514841133)

[6.1.3 Microsoft Visual Studio软件开发平台 42](#_Toc514841134)

[6.2 节点软件设计 42](#_Toc514841135)

[6.2.1 终端数据采集节点软件设计 42](#_Toc514841136)

[6.2.2 路由节点的软件设计 43](#_Toc514841137)

[6.2.3 协调器节点的软件设计 44](#_Toc514841138)

[6.3 监控PC端软件设计 45](#_Toc514841139)

[6.3.1 串口通讯程序 46](#_Toc514841140)

[6.3.2 阈值检测算法程序 47](#_Toc514841141)

[6.3.3 遗传算法程序 48](#_Toc514841142)

[6.3.4 BP神经网络算法程序 49](#_Toc514841143)

[6.3.5 模糊逻辑算法程序 51](#_Toc514841144)

[6.4 软件整体工作流程 52](#_Toc514841145)

[6.5 本章小结 53](#_Toc514841146)

[7 模拟与实验结果分析 54](#_Toc514841147)

[7.1 实验环境搭建 54](#_Toc514841148)

[7.2 实验数据结果及分析 55](#_Toc514841149)

[7.3 本章小结 59](#_Toc514841150)

[8 结论及展望 60](#_Toc514841151)

[8.1 总结 60](#_Toc514841152)

[8.2 展望 60](#_Toc514841153)

[致谢 62](#_Toc514841154)

[参考文献 63](#_Toc514841155)

[附录A 节点软件设计 64](#_Toc514841156)

[附录B 监控软件设计 67](#_Toc514841157)

# 1 绪论

## 1.1 论文的研究背景及意义

进入21世纪之后，全世界每年因火灾事故造成直接死亡人数约为7万人，造成的经济损失更是不计其数。火灾是不可避免的自然灾害之一，火灾会对人民群众的生命、财产、健康、安全等造成巨大的损失。

2017年1月至10月，国内共接报火灾约21万起，伤亡人数共计1700余人，经核对，火灾所造成的直接财产损失约为26亿元[1]。据统计，发生较多火灾地点为居民住宅区，约占火灾总数的43.5%，死亡人数为821人次，占死亡人总数的77.1%，另有446人次受伤，占受伤人总数的65.7%[1]。

在统计数据中发现一个规律，在越发达的国家，因火灾造成的死亡人口与经济损失相对较低，而在发展中国家相对较高。

在火灾发生的时候，一般第一时间内并不能采取对应的解决办法，不能防患于未然，往往是等到火灾火势逐渐到达控制不了的地步才进行报警，在此期间消防人员出动也需要一定的时间，一旦火势达到已经不能控制的局面，尤其是发生在楼宇的情况下，事故后果将会十分严重，也会造成非常大的经济损失。如何在火灾将要发生的时候、火灾火势在自控范围内的时候及时发现火灾的存在，并采取相应的措施防止火灾进一步扩大，将火灾扼杀在摇篮中，这种预警方式就显得格外重要了。

就目前市面上的大多数预警系统所面临的问题主要在两个方面体现：一是采用的是有线方式进行连接传输的，这种预警方式由于需要大范围的安装通信材料，安装较为麻烦，且成本高、工程量大等缺点，所以并没有得到广泛的应用。因此研究无线楼宇火灾预警系统对提高人们的生活质量有着不可小觑的作用；二是传统的火灾报警器几乎没有对多个传感器进行数据融合算法的处理，导致精度达不到要求，且会产生一定的误报情况，解决其精度，减少误报率是一个成熟的火灾报警器必须克服的问题。

本文针对传统设备的这一痛点，详细得介绍了如何对火灾探测器应用ZigBee技术进行无线传输和对所采集的数据进行数据融合。

## 1.2 国内外研究现状

### 1.2.1 国内研究现状

近年来，国内的火灾预警系统应用技术取得了相应的成就。但在实际运用中，由于技术发展较为晚，导致在通讯协议制定、精度等一系列问题上有一定的欠缺。例如：可靠性差而产生的误报；火灾探测器的算法单一导致的精度不高；在火灾智能报警与善后问题的技术上水平较为落后。主要体现在以下几种情况：

（1）标准规范不统一[2]。

国内已经针对各个地区，针对不同的情况已经颁布了一些消防规范和标准，但是由于不同的设备公司针对自己的产品并没有一个统一的设计标准，导致相同的设备可能换一个环境不能达到预期的效果，有时候则是需要在指定的公司进行相应的设备安装，这样使设备的兼容性大大降低，造成此类情况的主要原因还是由于标准不统一。

（2）报警系统的误报率和漏报率较高[2]。

此处的问题主要是归结于传感器的单方面输入。比如温感、烟感探测器，这种仅仅是单纯的阈值判断来对火灾进行报警，并不能排除偶然的室内温度过高、由粉尘引起的烟雾浓度增高、无烟雾火灾（例如酒精）的情况，导致火灾的误报率极高，智能程度不高。因为传统的火灾探测器的灵敏度并不是处于一个最优的状态，过低或过高的灵敏度分别会引起探测器的漏报或者误报。

（3）报警系统的适用范围小[2]。

由于我国的火灾探测器行业起步较晚，相应的政策规范颁布不全面，导致火灾探测器的安装范围仅仅局限于大型商场，人群密集的地方，忽略了人们的起居住所，同时，起居住所也是最易发生火灾的地方。

（4）火灾报警系统的构件连接需要改进[2]。

目前我国的大部分火灾探测器都采用的传统总线制连线方式，这种传统的布线方式存在很多缺陷，例如线与线之间存在干扰导致抗干扰能力差、导线发热存在精度不高等，这一系列情况都会导致火灾探测器的可靠程度低，在另一方面上，传统的布线方式耗费的人力物力将会很大，繁琐的布线方式也会对环境要求特别高。如果采用无线传输方式，就不需要考虑以上的问题，且在经济层面上也有了很大的节省，最主要的是提升了火灾探测器的精度。

（5）火灾报警系统的数据处理有待提高。

在国内，仅只有极个别的火灾探测器在智能化设计上做了相应的文章。总体来说，国内的大部分系统存在的问题主要是监测参数少，软件开发不成熟，存在较多BUG，算法欠缺等。传统的火灾探测器并未涉及多传感器数据融合处理，导致判断单一，偶然性大，系统的可靠程度不高。

### 1.2.2 国外研究现状

国外的火灾警报产品相对于国内来说起步较早，技术已经相当成熟了，在很多发达国家已经是实现了具有一系列的火灾防范措施，从火灾预防、报警、扑救，善后等一系列的事项都考虑的相当周到。

例如在美、法、英、日等发达国家，在城市火灾自动报警系统方面的钻研有丰硕的阅历。不仅在技术上有很高的成就，并且也投入正常使用多年，也很少存在误报、漏报、迟报等技术问题，这样能将现场损失降到最低。

且在发达国家中，对于智能化火灾报警系统管理方面很规范，相关政府也会成立专门的机构，来维护系统的正常运行，通信机制正常等工作，来保证在发生火灾时不会因为系统的过失造成相应的损失。

## 1.3 论文的主要研究内容

本文的章节分布及研究的主要内容如下。

第1章是绪论，着重介绍了本文研究的背景以及研究的意义，从国内外火灾探测器的研究发展现状做了简要的分析，更加应征了本文的研究意义，并对本文的架构做了一个简单的梳理。

第2章主要引出无线传感器网络的理论基础，然后根据无线传感器网络介绍了ZigBee无线通讯技术，为后面的系统总体设计方案提供理论依据。

第3章主要介绍了数据融合的处理方法，着重介绍了遗传算法（Genetic Algorithm）、BP神经网络算法（BP Neural Network Algorithm）与模糊逻辑算法（Fuzzy Logic Algorithm），接着对本系统的数据融合器总体方案设计进行了的规划，为后续的软件设计打下理论基础。

第4章确定了本系统的总体方案设计，包括硬件与软件的总体方案设计。

第5章主要介绍了系统的硬件方案与连接形式，主要包括主控制器（终端与协调器），传感器模块与串口通讯。

第6章主要确定软件设计的整体方案，重点研究了串口数据的接收，传感器采集的数据处理与信息融合算法的编程。

第7章对硬件系统进行仿真，经过制造实验火，对火灾探测器的终端与协调器之间的数据传输进行了验证，且数据经过BP神经网络机器学习后的更具有说服力，软硬件理论设计得以验证。

第8章是总结与展望，对本文所研究的系统进行了总结，并对系统提出了一些改善的建议与可以优化升级的扩展功能。

# 2 无线传感器网络技术及ZigBee简介

在第一章中，对传统无线火灾探测器的有线传输方式在实际应用中的弊端做了简要的分析。为了提高在实际火灾情景中系统的可靠程度，避免造成不必要的后果与财产损失，已有科学依据提出了无线传感器网络的应用。这一技术的提出，将无线传感器网络应用于火灾探测系统，能够很好的解决传统系统的痛点，这样不仅避免了现场布线的繁琐工序，也大大节省了人力财力，且现场监测的节点个数几乎不受限制。而对于一个庞大的无线传感器网络，需要构建一个能够承载这个网络的传输协议，ZigBee的技术很好地满足了这些要求。

本章将对无线传感器网络技术与ZigBee技术做具体的分析。

## 2.1 无线传感器网络技术

无线传感器网络（Wireless Sensor Network, WSN）所部署的区域内，有很多探测节点，节点与节点之间的通信方式不同于传统的有线通信，而是依靠无线方式来进行数据传输，节点之间以自组织和多跳网络模式进行工作。因此无线通信的方式有效地避免了传统设备的布线工序，并且能够全面覆盖监测区域，准确的采集现场数据并传输。

### 2.1.1 无线传感器网络体系结构

WSN的主要架构由三类节点组成，它们分别是传感器终端节点、路由节点和协调器（主控）节点[2]。由散布在现场监测区域内的大批传感器终端节点采集现场所监测的数据，依靠路由节点（中间节点），在终端节点与协调器节点之间构建通信桥梁，负责两类节点的通讯，从而扩大了监测范围与网络规模，而协调器则是为所有的终端节点加入其网络时调配一个独有的网络地址，以独特的短地址的通信方式接受各个网络所监测的数据而不会造成数据紊乱的情况发生。

图2.1为无线传感器网络的典型结构[2]。



图2.1 无线传感器网络的典型结构

Fig.2.1 Wireless Sensor Networks

### 2.1.2 无线传感器网络的特点

WSN具有如下几点特点：

1．传感器节点分布密集，规模庞大。

散布在火灾现场监测区域内的传感器采集节点可部署的规模是可调控的，根据现场环境、精度需求及其他，可以无限扩大规模，从而来保障整个系统在工作时不漏掉任何一个监测点。

2．硬件设计简单。

由于传感器节点网络仅仅需要采集数据，所以运用电路十分简单，且考虑到机器学习的过程在一般的MCU芯片上运行速率不能够满足要求，因此监控软件一般运行在PC机上。所以传感器节点做的事情极为精简。

3．自组织网络与多跳路由。

移动自组织(Ad Hoc)网络是一个多跳的临时自治系统。美国早在1968建立的ALOHA网络和1973年底提出的分组无线电（Packet Radio）网络[3]，而Ad Hoc的原型是就是以上两组网络。通过自己构建的网络，可在没有基站或者是不需要依赖基站等的情况下进行终端设施之间的彼此通讯。由于各个终端节点间的通讯距离可能只有数百米，达不到真正规模庞大的网络需求，此时多跳网络能够很好的解决，所谓多跳网络，即是通过中间节点转发至下一个节点，完成整个网络的数据转发。

## 2.2 ZigBee简介

本系统采用的是短距离无线通讯技术[4]，所谓的短距离无线通信技术定义为收发双方在传输距离较短（通常定义在几十米）的情况下，经过无线进行数据传输的通信技术。

ZigBee技术主要用于短距离且功耗要求较低的设施之间进行数据传输，并对一些典型的有周期性的数据或者是间歇性采集数据等传输也同样适用。

ZigBee网络的设计初衷就是为解决短距离传输网络的构建。所以ZigBee的体积小，功耗低，成本低廉，且ZigBee可扩展到65535个无线数据传输模块，此网络对于火灾探测系统可以说是十分庞大的一个网络。依据ZigBee的此特点，能够很好得作为无线传感器网络的传输协议。

### 2.2.1 ZigBee技术特点

ZigBee技术具备以下特点，分别是：

1．低功耗。

由于休眠工作模式是ZigBee技术的一大特点，再加上ZigBee的发射功率低至1毫瓦，因此ZigBee设备的低功耗是有目共睹的。

2．低成本。

ZigBee的传输协议不含版税，低生产成本，因此ZigBee模块的市场价格已为广大公众所接受。

3．低延时。

ZigBee的模式切换（休眠模式与工作模式间的切换、设备搜索与连接等）时间非常短，通常在60ms的时间内即可完成设备的搜索，激活，接入。因此ZigBee技术已经满足并兼容很多无线控制应用。

4．网络庞大。

一个星形的ZigBee网络可以容纳254个终端和一个协调器[5]。且依据ZigBee技术的16位短地址定义，一个ZigBee理论上最多能够管理的网络节点个数多至65535，而且不仅局限于此，还可以根据64位IEEE地址进行扩展，因而ZigBee可容纳的网络是相当庞大的[4]。

5．可靠程度高。

ZigBee通过既定的策略可以很好地避免发送数据的冲突和竞争。MAC层定义的协议中每次数据收发传输都需要等待对方的确认信息，才能执行后续操作。如果在传输的进程中，数据传输失败则可以立即重新执行。

6．安全性能高。

ZigBee提供基于循环冗余校验（CRC）的数据包完整性检查功能。它支持身份验证并使用AES-128加密算法，每个应用程序可以灵活设置其安全属性[5]。

### 2.2.2 ZigBee协议栈架构

ZigBee协议的创建初期时使用的是七层协议，但由于设计初衷是为了构建出低功耗、高速率的传输结构，后来将其结构简化为四层协议。其分别为物理层(PHY)、媒体访问控制层(MAC)、网络层(NWK)及应用层(APL)[6]。Zigbee的协议栈架构如图2.2所示。



图2.2 ZigBee协议栈架构

Fig.2.2 The architecture of ZigBee protocol stack

图2.3为开放系统互连参考模型 (Open System Interconnect 简称OSI）与ZigBee协议栈的对比。



图2.3 OSI模型与ZigBee协议栈结构

Fig.2.3 OSI Model and ZigBee protocol stack structure diagram

ZigBee的协议栈组成部分的每级都有自身独有的构建模型，每层所负责的主要任务也各不相同，下面对四层结构进行简要的介绍。

1．物理层(PHY)

物理层是整个协议层中最基本的部分，它指定了所使用的频带，以及诸如编码、调制、扩频和频率调制等无线传输技术[6]；物理层维护了一个自身的数据库，管理从物理频道发送和接收的数据。如图2.4所示为物理层结构模型。



图2.4 物理层结构模型

Fig.2.4 The PHY layer structure diagram

2．媒体访问控制层(MAC)

MAC层的主要作用定义了无线信道的访问控制机制，即它规定每个设备依次根据所建立的规则使用信道[6]，从而不至于产生访问冲突的情况。在网络中能够维护管理多个设备进行信道资源访问。

IEEE 803.4.4的MAC层提供了信道接入（CSMA-CA和TDMA）、本地网络建立维护和同步、安全、可靠通信等[7]。

MAC层同样维护了一个属于该层相关的数据库（主要是子层协议的相关状态）。如图2.5所示为MAC层结构模型。



图2.5 MAC层结构模型

Fig.2.5 The MAC layer structure diagram

3．网络层(NWK)

ZigBee网络层相当于整个协议栈的大脑，核心功能是路由、寻址、网络的形成与维护。

网络层主要负责网络的管理，例如网络发现与网络形成、加入网络和离开网络、节点的增加与删除。

网络层之所以可以实现星型拓扑结构、树型拓扑结构和网状拓扑结构等不同的网络拓扑结构，是因为在网络层内部支持多跳网络形成算法，所以理论上，多种复杂网络在网络层都可以进行初始化，只要有相应的算法支持。如图2.6所示为网络层结构模型。



图2.6 网络层结构模型

Fig.2.6 The NTK layer structure diagram

4．应用层(APL)

除了用户定义的应用对象之外，主要还包括几个部分：一是应用支持子层（APS）；二是ZigBee设备对象（ZDO）；三是应用框架等部分。

应用支持子层的作用是提供一个安全的隔离带，让应用程序与网络层有各自的负责任务而不冗杂，在两层之间，通过数据实体（APSDE）和管理实体（APME）来进行通信任务，完成数据通信与配置操作等，从而来完成应用程序的操作。

而对于ZigBee设备对象，也就是ZDO来说，主要维护的任务是对协议栈进行初始化，准备一切的网络结构与配置等等。且ZDO除了以上初始化操作外，还负责设备与设备之间的通信任务，例如设备之间的发现与绑定等。



图2.7 应用支持子层结构模型

Fig.2.7 The APL layer structure diagram

### 2.2.3 ZigBee与其他无线标准的技术特点比较

在无线通信的领域，不仅仅只有ZigBee通信技术，还有其他很多无线通信技术，关于任何一种的通讯技术都有其不同的适用范畴及领域。表2.1为ZigBee与其他无线标准技术的各方面性能对比，根据结果表明，ZigBee技术较其他技术更适用于本系统。

表2.1 ZigBee与其他无线标准的对比

Table 2.1 ZigBee compared with several other wireless standards

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 规范 | 工作频段 | 速率（Kbps） | 功耗（mW） | 连接设备数 | 主要用途 |
| ZigBee | 868915MHz,  2.4GHz | 20,40,250 | 1~3mW | 216~264 | 家庭网络，传感器网络 |
| 红外 | 820hm | 0.521,4,16 | 4~8 mW | 2 | 可见范围内数据传输 |
| HomeRF | 2.4GHZ | 1,2 | 100 mW | 127 | 家庭无线局域网 |
| 蓝牙 | 2.4GHz | 1,2,3 | 1~100 mW | 7 | 个人网络 |
| RFID | 5.8GHz | 0.212 | 无需供电 | 2 | 超市，物流管理 |

## 2.3 本章小结

本章主要通过无线传感器网络的应用来引出ZigBee技术，此技术在后文为系统的总体方案设计提供了相应的理论基础。

首先分析无线传感器网络的原理，根据其庞大的网络节点需求，ZigBee技术被选为系统的通信协议。然后重点剖析ZigBee技术的协议栈组成架构及主要负责任务等，接着依据其应用范畴与其他无线技术的性能比较，最终决定ZigBee技术的各方面功能较其他技术更加适用于本系统。

# 3 火灾报警系统的数据处理方法

针对于传统的单传感器火灾探测器来讲，传感器的数据处理过于单一，而对于一个火灾现场来说，并不仅仅局限于阈值的“与”或“非”的处理问题，而且需要经过多方面的参考。由于在实际生活中，非火灾信号和电磁噪声可能也会导致传感器有一定的输出值，即在特定的情况下，这些输出可能大于预设阈值，导致系统产生误报，此时，多传感器数据的融合变得更加重要。在人工智能技术发展下，将人工智能技术运用到火灾探测器中是一个必然趋势，机器学习方法能使系统更精确的工作，将传感器数据进行融合，使系统精度更高，鲁棒性强。本章将从对相关算法介绍与火灾报警系统的数据融合器设计进行相关的介绍。

## 3.1 相关算法介绍

此部分主要介绍遗传算法（Genetic Algorithm）、神经网络算法（Neural network algorithm）和模糊逻辑算法（Fuzzy logic algorithm），三个算法将作为火灾报警系统数据融合器设计的理论基础。

### 3.1.1 遗传算法

3.1.1.1 遗传算法简介

遗传算法从传统意义上讲，也是进化算法的一种。这是一个近似模拟达尔文的生物进化理论的“自然选择和适者生存”过程的计算模型[8]，它模拟自然演化过程并搜寻最优解。

利用“袋鼠跳”的例子可以很好解释遗传算法。假设在珠穆朗玛峰下生活着一群袋鼠，由于山下的猎人捕杀行为，导致袋鼠被迫往山上迁移，种群中强大的个体迁移速度比其余较弱的个体快，也就避免了暂时的被猎杀，逐渐弱势群体被淘汰，经过种群繁殖，基因变异，优良基因的传承，依据“生存竞争，适者生存”规律，种群最终会找到一个适宜生存的地方。最后的种群必然是一个最优种群。

3.1.1.2 遗传算法运行步骤及组成

在此算法中，由4个部分组成，分别是：种族群体（Ethnic group）、种群个体（Individual of population）、个体基因（Individual gene）、染色体（Chromosome）。

由于遗传算法是近似模拟生物进化论的过程，按照达尔文生物进化论中的原则，环境适应能力较强的个体更有可能参与群体的繁殖。经过无数代的繁殖，最终的种群中必将存在最优个体，也是项目的最优解决方案。相反，适应度低的个体后代就会越来越少，在其中渐渐的被种群淘汰。个体在繁衍后代时，会发生交叉( Crossover ) ，变异 ( Mutation ) ，在此过程中，适应度( Fitness )低的个体会被淘汰，而适应度高的个体会越来越多。所谓适应度，实则为针对项目所描述的问题，根据既定表达式所计算出的。

遗传算法的算法核心思想是通过随机值产生函数初始化一个种群，对每一个个体的特征值进行编码（编码形式有二进制编码、实数编码、符号编码等），通过选择（通常为轮盘赌法）来完成 “天择”，被选择的个体将会在种群中繁衍后代，完成基因的交叉或变异操作来提升个体的适应度，当种群遗传的迭代繁殖代数达到阈值时，算法退出，停止繁殖，并在种群中寻找适应度最大的个体。

遗传算法的基本运行步骤如图3.1所示。



图3.1 遗传算法的基本运行步骤

Fig.3.1 Basic operation steps of genetic algorithm

遗传算法主要包括以下几个核心算法子模块：遗传编解码、适应度计算、遗传操作算子[8]。

1．遗传编码。

遗传编码是遗传算法进行中的前提。遗传算法工作时，首先会将特征值编码成可用于后续遗传算子的操作形式，在进行遗传操作结束后，也有与之相对应的解码函数来完成特征值的还原，以便于适应度的计算。迄今为止，编码形式有很多，例如二进制编码、实数编码、符号编码等。但二进制编码是最常用的编码形式，可直接被机器操作，理解简单。

2．适应度计算。

为了满足优胜劣汰的要求，会对每个个体进行适应度的计算用以评判是否可以继续存在于种群之中。

3．选择算子。

常用的选择算子有以下几种：排序选择、比例选择、轮盘赌法等。本系统采用轮盘赌法进行种群的一个选择保留。进行此操作的目的是模拟自然选择的过程，看似随机，实则必定会决定一个发展方向。此处选择算子有很多，常见的的轮盘赌法算法如图3.2所示，轮盘转起来，指针指向的区域则会被选中，当种群被选择算子填满后，停止选择操作。



图3.2 选择算子的工作流程

Fig.3.2 Select operator workflow

4．交叉算子。

交叉操作是模拟生物界中繁衍时的基因重组的过程，将父代的两部分基因进行交换重组，产生新的个体，以继承父类的优良基因。交叉操作包括单点交叉、多点交叉、算数交叉等多种交叉形式[8]。

如图3.3为本系统的交叉算子的功能示意。



图3.3 交叉算子的功能示意

Fig.3.3 Functional description of crossover operator

5．变异算子。

变异算子是模拟生物进化中基因突变的过程。在遗传算法中，当某个个体的某段特定的二进制编码位发生改变时，个体的适应度也会随之发生改变，从而可以在变异的过程中，保留优良个体的基因，从而使种群最终接近于最优解。

如图3.4为变异算子的功能示意。



图3.4 变异算子的功能示意

Fig.3.4 Function of mutation operator

### 3.1.2 神经网络算法

3.1.1.1 人工神经元模型

众所周知，人的大脑可以用来处理身体部位发送的复杂信息，根据经验和逻辑来推测行为并做出相应决策，并能对各种信息进行分类归纳，这就是生物神经上常说的“自学习”过程[9]。

如果我们将人脑的这种“自学习”过程运用到实际生活中，让机器自己进行学习和探索，模拟人脑建立对数据的信息处理的网络，于是便引出了人工神经网络。人工神经网络属于一种自学习的系统[9]。如图3.5所示的为单个人工神经元模型，通过机器自学习过程，建立一个满足要求的算法，函数，或者是一种规则。机器学习的过程其实很简单，是根据输入值经过前馈输出值与期望输出值进行对比然后进行反馈不断修正网络的阈值和权重，是一个自适应的过程，当前馈输出值与训练数据中的期望值两者的误差达到网络既定的要求时，则机器学习任务完成，人工神经网络建立完毕。



图3.5 人工神经元模型

Fig.3.5 Artificial neuron model

由图3.5可以得出，该单个人工神经元模型不仅只有一个输入信号，n个输入信号经过与权值的加权以及激活函数的处理，与阈值的矫正，最终输出决策值。该神经元各部分的关系如公式3-1和公式3-2所示。

 （3-1）

 （3-2）

 —— 输入信号；

 —— 处理单元阈值；

 —— 输入层i到j的权值。

f(x)为激活函数。针对神经网络算法，激活函数要求克服线性函数表达力低的缺陷，所以要求激活函数需要加入非线性因素，因此采用Sigmoid函数。Sigmoid函数公式如3-3所示。

 （3-3）

对于人工神经网络算法，人们已经提出十余种模型，每种模型都有其适合的场景。目前应用最为广泛的神经网络模型为前向神经网络模型，数据经过前向网络，最终达到输出层，在输入层与输出层之间可以有很多隐藏层，这样就构成了一个由输入层、隐藏层与输出层组成的一个复杂的神经网络。BP神经网络算法是前向神经网络算法的典型应用实例，以下着重介绍BP神经网络算法。

3.1.1.2 BP神经网络算法

BP（BackProgation）神经网络算法属于前向神经网络的代表[10]，其组成结构如图3.6所示。BP神经网络不同于简单神经网络的是其输入学习样本后，将得到的输出值与期望值进行比较可以得到相应的误差值，根据误差值可以推算出影响误差产生的原因，主要是权值与阈值导致的，经过反馈处理来不断的修正阈值与权值，直到输出结果满足预期的要求则终止学习。



图3.6 BP神经网络算法结构图

Fig.3.6 BP neural network algorithm structure diagram

BP神经网络算法同前向神经网络算法一样，拥有三层结果，其中输入层与输出层仅有一层，且节点个数根据训练样本和输出期望而定，而隐藏层数量与隐藏层节点数不限，根据用户的需要自己设定。理论上来讲，如果训练数据与隐藏层的层数与节点数足够大，系统能够模拟出任何方程。一般来讲，节点数可以根据以下经验公式来进行隐藏层节点数的确定：

 （3-4）

 —— 隐藏层节点数；

 —— 输入层节点数；

 —— 输出层节点数；

 —— 1~10之间的整数。

当隐藏层节点数确定之后，整个网络基本上就确定。

当输入的样本数据通过公式3-1的计算与3-2的激活，输入层的输出数据作为隐藏层的值数据，然后将隐藏层的值同样经过公式3-1的计算与3-2的激活，将经过隐藏层计算出的值作为输出层的值，即为网络的预测值。

此时仅仅完成的前向输出工作，BP神经网络训练方法其独特之处在于会通过输出层的值与期望值之间的误差对阈值与权值进行一个修正，最终使系统的误差在许可的范围之内。

根据误差值（Err）反向传送进行修正步骤如下。

首先计算输出层预测值与期望值两者之间的误差值：

 （3-5）

 —— 输出层误差值；

 —— 输出层实际值；

 —— 输出层期望值。

其次计算隐藏层的误差：

 （3-6）

 —— 隐藏层误差值；

 —— 输出层误差值；

 —— 输出层实际值。

误差值计算完成之后，需要对网络各层之间的权重和节点固有阈值进行更新：

 （3-7）

 （3-8）

 （3-9）

 （3-10）

 —— i节点到j节点（输入层到隐藏层或隐藏层到输出层）的权重更新值；

 —— 当前节点的阈值更新值；

 —— 神经网络学习速率；

 —— 当前节点的误差值；

 —— 输出层的实际输出值；

经过以上步骤反复训练，最终达到实际输出与期望值输出之间的误差在预设误差内即可完成网络训练，并将这些参数（权值与阈值）作为最终网络的参数。

图3.7为BP神经网络算法流程图。



图3.7 BP神经网络算法流程图

Fig.3.7 BP neural network algorithm flow chart

### 3.1.3 模糊逻辑算法

模糊逻辑是建立在多逻辑决策值推理原理的基础上，模拟人脑思维的一个模式，来解决一个需要多方位判断的模糊规则，实行模糊综合判断。模糊逻辑处理流程如图3.8所示.



图3.8 模糊化流程

Fig.3.8 Folw of fuzzy

模糊逻辑算法需要对输入的特征信息进行一个量化，确定他们的论域。在无特殊情况下，论域U上的模糊集分为七级，分别是负大（NB）、负中（NM）、负小（NS）、零（ZD）、正小（PS）、正中（PM）、正大（PB）[2]。根据隶属度函数（例如三角隶属度函数，正态隶属度函数等）确定每一个特征信息属于哪一个层次。

判断特征值所属论域后，则可建立控制规则表。举例说明，当A成立，B成立，则C成立；或者是当A是PB，B是NB时，C不成立。规则的表现形式根据用户的需求，可以有多条，数目由输入和量化等级决定。

当特征值经过隶属度函数处理之后，最终都会根据既定的表来查找对应的决策值。

## 3.2 信息融合技术

信息融合（Data Fusion）技术也称为多传感器信息融合技术或数据融合技术[4]。信息融合是一个对从单个和多个信息源获取的数据和信息进行关联和综合，以获得精确的位置和身份估计，以及对态势和威胁及其重要程度进行全面评估的信息处理过程[11]。在本系统中，根据集成传感器模块所采集到的不同的数据信息，以既定的准则加以剖析，将多种数据最终融合在一起，输出一个最终的决策值，这样克服了传统单传感器的缺点，使系统更优越，性能更高。

如图3.9为传感器数据融合关系结构，总体可以分为三层：数据层融合、特征层融合与决策层融合。



图3.9 传感器数据融合技术层次关系

Fig.3.9 Sensor Data Fusion Technology Hierarchy

1．数据层融合。

该层数据处理一般在传感器所采集到的原始数据层。主要负责的任务是对数据进行预处理。数据层在信息融合技术层次里的等级属于最低级，该层任务是尽可能多得保存现场真实数据，方便后面两层的数据处理。

2．特征层融合。

对现场所采集的原始数据进行一定的预处理，提取各个数据的特征后，针对这些特征进行分析，由此可见，相比数据层融合，该层的层次高于数据层，同时特征层也显得更加抽象。特征层融合分为两大类：一是特征层融合的目标状态融合；二是特征层融合的目标特性融合[4]。前者的状态融合主要负责对数据的预处理，然后实现各参数的状态转换；后者利用模式识别相关技术完成特征值的提取与转换。特征层融合后的数据较原始数据相比进行了压缩，同时保留原始数据的有用信息。通过本层融合后的数据，便于后一层的的提取与融合数据的需要。

3．决策层融合。

决策层融合经过前两层融合所提取的特征向量值，根据既定准则和决策的可信度做出最终的判断。决策层是最高的层次，结合特征层的数据输出，对问题做出最终判断。

## 3.3 信息融合技术在火灾报警系统中的应用

根据国内目前火灾探测器的发展情况，传统的火灾探测器对现场的数据采集处理十分简单，没有复杂的处理逻辑，导致系统的可靠程度不高，缺陷较为明显。本系统所研究的是一个高性能，高可靠性，高精度的火灾探测器，因此不仅局限于无线传感器网络的数据传输，还需要对采集的数据进行融合处理。

据实验结果表明，大部分的现场火灾发生时，温度、湿度、O2浓度、CO浓度、烟雾浓度等均会发生变化。但在变化过程中，温度，CO浓度与烟雾浓度的变化速率显著，因此这三个参数是最能反映火灾发生的检测对象，依据信息融合理论基础，本系统的火灾探测融合过程依旧采用三层结构，在3.2节中已做详细介绍。如图3.10所示设计了复合火灾探测器的数据融合器。

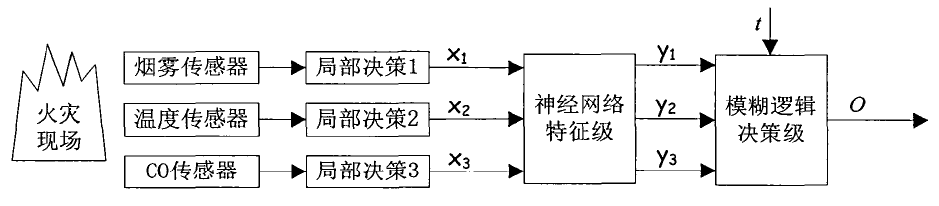


图3.10 复合火灾探测器

Fig.3.10 Composite fire detector

以下对本火灾报警系统的数据融合器进行分层次介绍。

### 3.3.1 数据层融合器

数据层融合发生在集成传感器，如图3.10所示为局部决策。传感器采集现场数据后，经过简单的判断处理，初步确定是否有火灾发生的可能性，如果有发生火灾的可能性，则将数据保留，进行下一步处理，反之，系统将继续采集工作不做任何处理。

本系统在数据层融合器中主要运用到的处理方法为阈值检测算法。火灾探测器从现场所采集到的原始数据并不是所有的数据都直接用于后期处理判断，在当前层会将数据进行预处理。阈值检测法为最为直观的预处理算法，若阈值多次超过预设值时，则判定火灾有可能发生，然后将该组数据传入下一层进行进一步融合。

阈值检测算法如公式3-11与公式3-12所示：

 （3-11）

—— 单位阶跃函数；

—— 第i个传感器的阈值；

—— 第i个传感器的第k次采集到的数据；

—— 第i个传感器第k次的决策结果。

 （3-12）

m —— 为采集次数。

当（）任意一个输出为1时，说明火灾探测器的复合传感器模块中所采集的三个信号（温度，CO浓度，烟雾浓度）有超过预设阈值，此时输出结果是可能发生火灾，此时保留数据，传入下一层进行决策判定。

### 3.3.2 特征层融合器

本系统采用基于遗传算法优化的BP神经网络算法作为火灾报警系统的特征层融合器。如图3.11所示为基于遗传算法的BP神经网络算法。

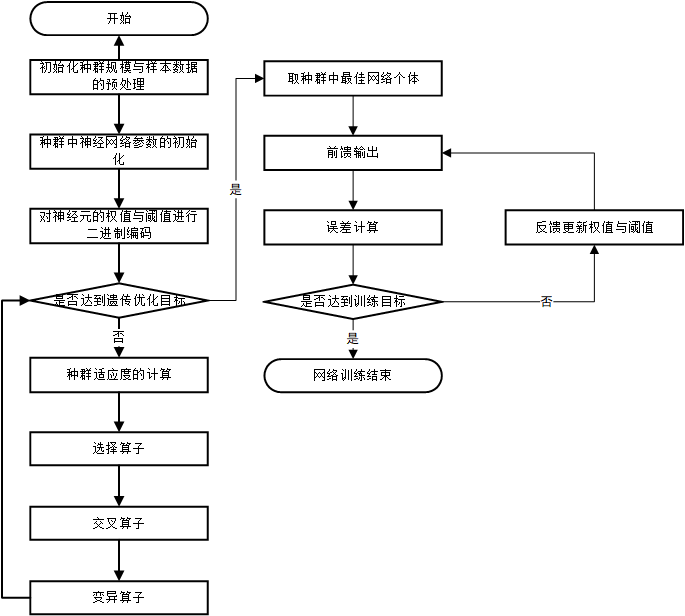


图3.11 基于遗传算法的BP神经网络算法流程图

Fig.3.11 BP neural network algorithm flow chart based on genetic algorithm

将经过遗传算法选择后的最优网络个体作为训练网络的初始值，将数据层融合器的火灾特征参数进行归一化处理。如图3.10所示为神经网络特征级。分别用来表示温度传感器输出参数，烟雾传感器输出参数与CO传感器输出参数。三类参数输入到特征层后，经过训练完成后的网络进行数据的计算和处理，判断火灾处于非火灾，阴火还是明火状态，其结果分别用来表示。如图3.12所示为特征层融合器结构框图。

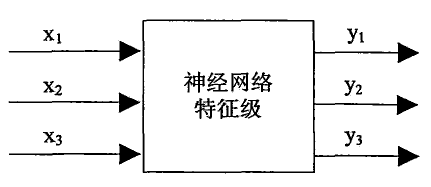


图3.12 特征层融合器结构框图

Fig.3.12 Feature layer fuser block diagram

根据3.1.2节所介绍的BP神经网络算法，本系统针对此算法确定了系统网络的设定。由于隐藏层的节点数对系统的性能影响较大，如果网络过于庞大，导致系统的收敛速度过慢，学习时间过长；如果网络过于简单，可能会导致系统受干扰比较大，使系统的精度不高。经过多次实验和理论依据，本系统设定为1层隐藏层，隐藏层的节点数为7个。火灾报警系统的BP神经网络如图3.13所示。



图3.13 火灾报警系统的BP神经网络

Fig.3.13 BP Neural network of fire alarm system

对于不同环境的火灾有不同的输入数据，火灾的数据量就十分庞大，此时只针对重要的情况进行数据训练，其他情况我们并不关心。本系统数据融合训练数据选择中国明火SH4和标准阴燃火SH1的相关数据。标准数据如表3-1所示。

表3-1 BP神经网络算法训练数据

Table.6.1 BP Neural network algorithm training data group

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 温度 | 烟雾 | CO气体 | 非火灾概率 | 阴燃火概率 | 明火概率 |
| 1 | 1.000 | 0.200 | 0.900 | 0.000 | 0.050 | 0.950 |
| 2 | 0.900 | 0.230 | 1.000 | 0.000 | 0.050 | 0.950 |
| 3 | 1.000 | 0.220 | 0.850 | 0.020 | 0.080 | 0.900 |
| 4 | 0.900 | 0.220 | 1.000 | 0.020 | 0.080 | 0.900 |
| 5 | 0.875 | 0.180 | 0.350 | 0.050 | 0.100 | 0.850 |
| 6 | 0.950 | 0.200 | 0.750 | 0.030 | 0.100 | 0.850 |
| 7 | 0.900 | 0.130 | 0.200 | 0.100 | 0.100 | 0.800 |
| 8 | 0.925 | 0.180 | 0.250 | 0.100 | 0.100 | 0.800 |
| 9 | 0.600 | 1.000 | 0.750 | 0.700 | 0.100 | 0.200 |
| 10 | 1.000 | 1.000 | 0.750 | 0.700 | 0.100 | 0.200 |
| 11 | 0.450 | 0.150 | 0.500 | 0.050 | 0.800 | 0.150 |
| 12 | 0.450 | 0.150 | 0.700 | 0.050 | 0.850 | 0.100 |
| 13 | 0.450 | 0.180 | 0.750 | 0.050 | 0.850 | 0.100 |
| 14 | 0.600 | 0.700 | 0.450 | 0.800 | 0.100 | 0.100 |
| 15 | 1.000 | 0.420 | 0.700 | 0.850 | 0.050 | 0.100 |
| 16 | 0.500 | 0.250 | 0.700 | 0.020 | 0.900 | 0.080 |
| 17 | 0.525 | 0.500 | 0.750 | 0.020 | 0.900 | 0.080 |
| 18 | 0.550 | 0.830 | 0.750 | 0.000 | 0.950 | 0.050 |
| 19 | 0.375 | 0.370 | 0.450 | 0.850 | 0.100 | 0.050 |
| 20 | 0.625 | 0.200 | 0.500 | 0.850 | 0.100 | 0.050 |

在输入训练数据准备完毕之后，则可以根据3.1.2节所介绍的BP神经网络的工作流程开始机器学习的过程，最终启动属于本火灾报警系统的神经网络（具体软件实现在第六章介绍）。

### 3.3.3 决策层融合器

决策层融合器主要是应用模糊逻辑技术，在本章第3.1.3节已经对模糊逻辑技术的理论做了简要的介绍，本小节针对其在火灾报警系统中的应用再次做一个详细的介绍。

将明火概率与阴燃火概率进行量化，确认其所属论域与模糊等级。非火灾概率不是本系统所关心的概率，至此，我们需要考虑系统的智能化程度，增添一个控制规则为危险信号持续时间，同样对时间信号进行量化。

图3.14所示为决策层融合器结构。



图3.14 决策层融合器结构

Fig.3.14 Decision layer fuser structure

y1 —— 明火概率；

y2 —— 阴燃火概率；

T —— 危险信号持续时间；

u —— 决策层输出结果。

在决策层，主要任务是将三个输入数据利用正态分布隶属度函数进行模糊化成为PB、PM、PS三个等级（由于概率不可能为负，零可以近似为极小值）。正态分布隶属度函数如公式3-13所示。

 （3-13）

式中b根据经验要求取值0.2，a根据模糊规则中PS、PB、PM分别取值0,0.5,1；

本系统我们做一个特殊处理，将火灾持续时间模糊化为两个等级（PB与PS），且阴火火灾概率与明火火灾概率不可能同为PB或者PM与PB同时存在，因此，火灾模糊逻辑规则如表3.2所示。

表3.2 模糊推理规则

Table.3.2 Fuzzy Inference Rules

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | T | u |
| PS | PS | PS | PS |
| PS | PM | PS | PS |
| PS | PB | PS | PS |
| PS | PS | PB | PS |
| PS | PM | PB | PB |
| PS | PB | PB | PB |
| PM | PS | PS | PS |
| PM | PM | PS | PB |
| PM | PS | PB | PB |
| PM | PM | PB | PB |
| PB | PS | PS | PS |
| PB | PS | PB | PB |

## 3.4 本章小结

本章首先着重介绍了三大算法，为信息融合技术在火灾报警系统中的实现提供了理论依据；其次简要介绍信息融合技术；根据系统的要求与前面的理论基础，最后对系统的数据融合器进行了相关设计。依据信息融合技术的理论，本系统的火灾数据融合器分为数据层融合器、特征层融合器和决策层融合器三个部分。针对每层完成的任务各不相同，数据层是最低级的一层，通过阈值检测算法对采集的原始数据进行预处理与预判断；特征层则经过训练好的神经网络来自行运算与处理；决策层依赖模糊逻辑算法与相应的模糊规则对明火，阴火概率以及危险信号持续时间进行最终的决策判断，输出最终的火灾情况。

本系统经过数据融合处理后的输出决策值来判定火灾的最终状态，能够解决传统设备的不足之处，改善系统的性能和可靠性。

# 4 火灾报警系统的总体方案设计

## 4.1 系统设计原则

从系统的应用场景出发，火灾报警系统要求时效性强，可靠性高，稳定强度高，抗干扰能力强等。

系统的硬件在设计过程中，需要对每一个传感器进行严格选型，选型依据是传感器的精度和适用场景，同时成本因素也应考虑进来。同时，为后期系统的优化升级与发展，系统需要预留一些扩展的I/O口供设备的升级。

系统的软件设计则是分模块设计，将每个模块结构化，然后统一编写与联调，这样可以提升开发效率，也对代码复用有一定的帮助。

## 4.2 选型分析

### 4.2.1 处理器选型

4.2.1.1处理器选型

目前市面上有很多ZigBee技术的无线芯片与SOC产品，如何选用最适合本系统的芯片则显得额外重要。如表4.1所示为常见ZigBee芯片的性能比较。

表4.1 常见ZigBee芯片性能比较

Table.4.1 Common ZigBee chip performance comparison

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 各芯片型号 | JN5148 | CC2530 | MC13192 | EM260 |
| 工作频率（Hz） | 2.4G | 2.4G | 2.4G | 2.4G |
| 无线速率（Kbit/s） | 250 | 250 | 250 | 250 |
| 发射功率（dBm） | +2.5 | +4.5 | +3.6 | +3 |
| 接收灵敏度（dBm） | -97 | -97 | -92 | -97 |
| 最大发射电流（mA） | 15 | 35 | 35 | 37.5 |
| 最大接收电流（mA） | 18 | 24 | 42 | 41.5 |
| 休眠电流（uA） | 0.2 | 1 | 1 | 1 |
| 工作电压范围（V） | 2.0~3.6 | 2.0~3.6 | 2.0~3.4 | 2.1~3.6 |
| 硬件开放度 | 不开放 | 开放 | 开放 | 开放 |

除了要满足火灾数据采集的各方面要求，还需要从芯片的可靠性，性价比与系统的开放程度多方面入手，综合考虑最优方案[12]。

经过对表4.1中各芯片的综合对比，且考虑性价比最高的选择，本系统采用TI（Texas Instruments, 德州仪器）公司的CC2530芯片作为本系统的处理器。

4.2.1.2 CC2530芯片简介

此款芯片很好的兼容了德州仪器公司所提供的ZStack协议栈，因为ZStack协议栈能与开发工具很好的结合起来，使得用户在开发过程中能更加便捷。

### 4.2.2 节点传感器选型

众所周知，对于一个优良的测量系统，传感器的优良与工作效率往往决定了系统的性能，因此，对于传感器的选择是非常重要的。在前面第三章中所介绍的数据融合处理方法中提到，系统采用温度、烟雾浓度与CO浓度为监测对象。下面分别对三个参数所用到的传感器进行分析。

1．温度传感器。

本系统的温度传感器采用DS18B20。DS18B20具有小体积、不易受外界干扰和测量分辨率低等特点，且为数字信号输出，不需要相应的的AD转换器既可以直接输入到微处理器内。

DS18B20的测温范围最低为－55℃，最高为+125℃，且测温误差在1℃左右，完全满足火灾测量的温度需求。另该传感器的接线方式简单，一根线将DS18B20与处理器相连即能完成两者的双向通信，且测量结果不需要经后续电路的处理，直接为9~12位数字量方式串行传送。

2．烟雾传感器。

本系统使用的烟雾传感器为MQ-2气敏传感器。

该器件的阻值与气体浓度之间一般呈非线性关系，但在标准浓度域的气体检测时可近似认为是线性的。因此，一定的烟雾浓度范围内，检测是有效的，它不适合于仪表之类精确浓度的测定，但是完全可以满足本系统对烟雾浓度采集的要求。

MQ-2型传感器对可燃性烟雾气体的灵敏度极高。其工作原理主要是源于其内部组成材料为二氧化锡半导体气敏材料，当火灾发生时，空气中的烟雾与二氧化锡结合会导致电阻值发生改变，使输出的模拟信号发生变化，从而可以监测空气中烟雾浓度。

3．CO传感器。

本系统选用的CO传感器为MQ系列的MQ-7 CO传感器，此传感器的工作模式如同上所述的MQ-2烟雾传感器，传感器的输出大小会随空气中一氧化碳气体浓度的变化而变化，且变化呈正比变化，因此所监测的空气中CO浓度变化满足本系统的要求。

## 4.3 系统总体方案设计

### 4.3.1 系统硬件总体架构

楼宇火灾报警系统的硬件总体架构包含五个部分：火灾探测复合传感器模块、发射无线通信节点、路由器、接收无线通信节点与主控制器。图4.7为系统硬件总体架构。



图4.7 系统硬件总体架构

Fig.4.7 System hardware architecture

根据4.2.2的传感器节点选型，探测器节点拥有三个传感器，分别是DS18B20、MQ-2与MQ-7，利用三个传感器来收集现场的三个监测参数（温度，可燃性气体浓度与CO浓度）。本系统中的DS18B20为数字信号，可以直接输入使用，而MQ-2与MQ-7则需要进行AD转换将物理信号转换为电信号再进行后续处理。当探测器节点采集数据完成之后，依靠ZigBee技术将数据发送到主控制器进行监测软件上的数据融合处理。

无线传感器网络与ZigBee技术的应用是实现本系统的技术支持，没有数据交互也就不存在监控的概念，整个网络负责数据的收发与通信，ZigBee技术负责管理网络节点，使系统有序的进行工作。整个硬件系统架构缺一不可，而探测器又如同整个系统的感觉器官，也是最基础的。

### 4.3.2 系统软件总体架构

本系统最重要的通信网络是利用ZigBee技术来构成的，其主要包括三个部分：终端节点（探测器节点）、路由器节点与主协调器节点三部分构成。无线网络由协调器负责建立，然后路由器节点和终端节点通过扫描通信范围内的可用信道向协调器申请加入网络，在无特殊情况下，协调器同意网络加入后为其两个子网络分配一个独有的短通信地址，从而建立组网[2]。

各节点的编程任务如下：

终端节点负责对信号进行编码，对编码后的字符合法性也需要进行判断；同时终端节点负责数据的传输。

协调器不仅要负责网络的形成，且还需要对终端节点数据的收发进行负责。

本课题的CC2530硬件编程使用IAR嵌入式软件开发平台作为开发环境。无线传感器网络的初始化程序需要使用TI提供的协议栈ZStack-CC2530来进行开发。

通讯方式建立完成之后，我们需要对CC2530的串口通讯进行编程，也就是如何在协调器与PC端建立相应的通讯机制。

PC端接收到数据之后，在PC端主要是做数据融合处理，首先需要对神经网络完成建立与训练，本课题在第三章中介绍了如何初始化一个基于遗传算法优化的BP神经网络算法，此网络相当于整个PC端软件的心脏，实时对协调器传输的数据进行一个预测。数据融合过程首先是通过数据层融合器，利用阈值算法来确认是否可能是发生火灾，如果被确定疑似火灾数据则进行下一步的特征层融合，特征层融合器通过服务器启动时初始化的BP神经网络，经过神经网络的前馈运算后，最终送到决策层融合器进行模糊逻辑算法的一个决策值输出，最终判断是否为火灾发生。

软件编程要求具有模块化思想，各部分有各部分的特有的功能。图4.8为系统软件设计的总体架构。

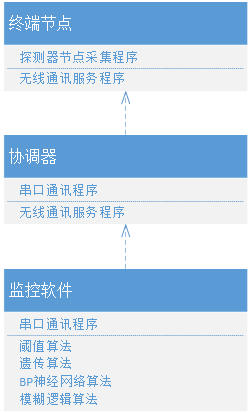


图4.8 系统软件总体架构

Fig.4.8 System Software Architecture

## 4.4 本章小结

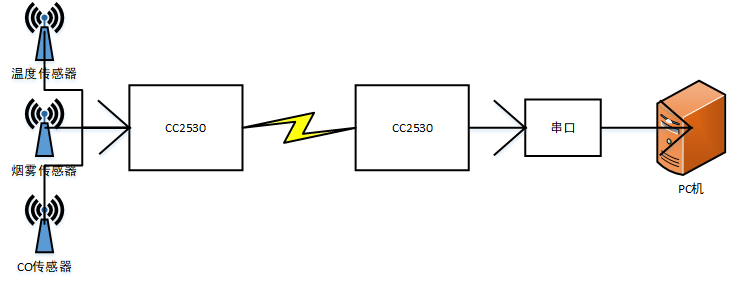
本章确定了本火灾探测器的设计原则，然后根据系统的要求，结合实际情况，对硬件进行选型，分别确定了系统的主控制器使用TI公司最新的ZigBee片上解决方案CC2530，探测器节点使用DS18B20、MQ-2与MQ-7分别来监测温度、烟雾浓度与CO浓度。在本章的最后，提出了系统的硬件与软件设计总体架构，为后续章节的具体设计做好准备。

# 5 火灾报警系统的硬件设计

本章是在火灾报警系统的硬件总体架构的基础上进行细化设计。将硬件的设计分为两个部分，分别是终端电路与协调器电路。

## 5.1 总体硬件设计思路

在图4.7中已经给出系统的硬件框架。如图5.1所示提出针对本系统具体的无线通信节点硬件结构图。



终端节点

协调器节点

图5.1 无线通信节点硬件结构图

Fig.5.1 Wireless communication node hardware structure diagram

如图所示，本系统的硬件结构较为简单，主要负责将现场所采集到的数据从终端节点转发给协调器，最终通过协调器中集成的串口通讯电路转发给PC机进行数据融合处理。到此，硬件电路的设计工作完成。

## 5.2 CC2530最小系统硬件电路

CC2530内部集成了一个8051内核CPU与一个RF射频发射模块，其内部也集成了大容量存储器。

如图5.2为CC2530最小系统电路图。

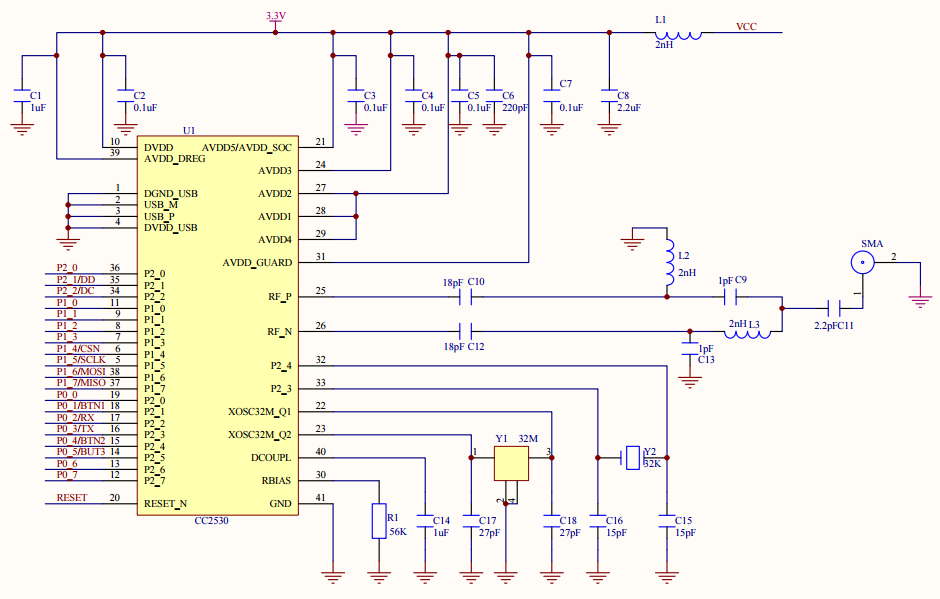


图5.2 CC2530最小系统电路图

Fig.5.2 CC2530 chip peripheral circuit diagram

CC2530芯片内部详细参数与系统特性如图5.3所示。



系统特性

系统组成

图5.3 CC2530芯片内部详细参数

Fig.5.3 CC2530 chip internal detailed parameters

CC2530中P1口与P2口作为普通IO口使用，本系统使用的I/O口为P0\_5、P0\_6与P0\_7所示。

## 5.3 终端节点硬件电路设计

### 5.3.1 温度传感器节点

DS18B20引脚图4.3所示。



图5.4 DS18B20引脚图

Fig.5.4 DS18B20 Pin Diagram

由上图可知，我们只关心三个引脚，GND接入CC2530公共地线，VDD端通过电源模块提供3.3V直流电。I/O口接入CC2530的P0\_7口进行数据的传输。

DS18B20与CC2530最小系统的连线图如下图所示。

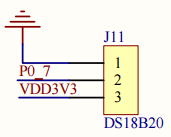


图5.5 DS18B20连线图

Fig.5.5 DS18B20 Connection diagram

### 5.3.2 烟雾传感器节点

MQ系传感器应用电路图如图5.6所示。

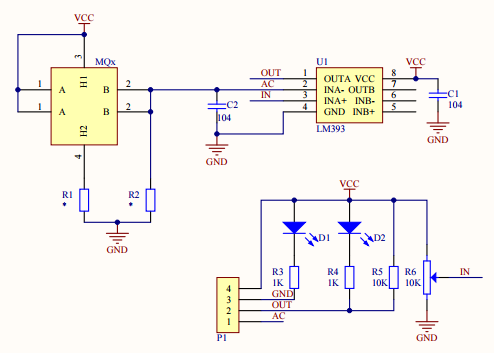


图5.6 MQ系传感器应用电路图

Fig.5.6 MQ series sensor application circuit diagram

MQ-2烟雾传感器的特性如图5.7所示。

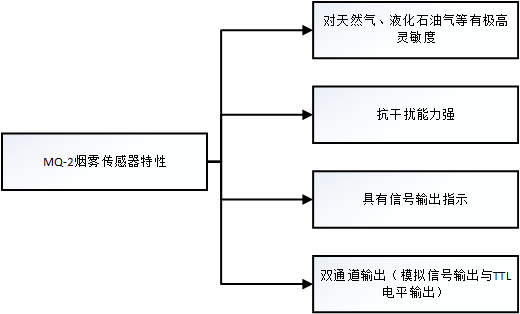


图5.7 MQ-2烟雾传感器的特性

Fig.5.7 MQ-2 Smoke Sensor Features

对于MQ-2烟雾传感器来说，我们不需要关心其TTL输出电平，所以将DO引脚悬空或者接地。其余引脚连接方式将VCC连接3.3V，GND接入CC2530最小系统的公共地线，而AO模拟信号我们不能直接使用，需要经过AD转换成数字信号，此处AD转换是经过芯片的内部转换来转换成我们所需要的数据，AD转换在后面会继续介绍。MQ-2连线图如下所示。

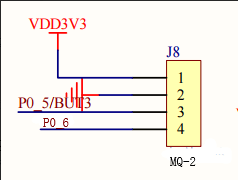


图5.8 MQ-2连线图

Fig.5.8 MQ-2 Connection diagram

### 5.3.3 一氧化碳探测器节点

图5.9为MQ-7的相关特性。

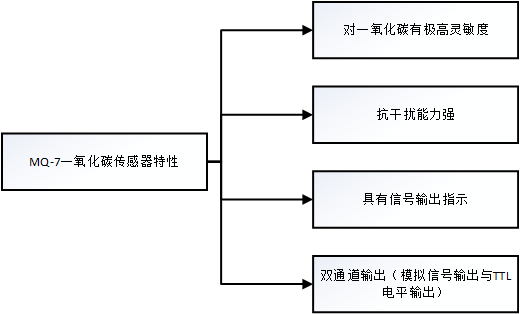


图5.9 MQ-7烟雾传感器的特性

Fig.5.9 MQ-7 Smoke Sensor Features

由于MQ系传感器除监测对象不同外，其余的使用方式完全可以参照MQ-2的应用方式，本系统MQ-7的连线方式如图5.10所示。

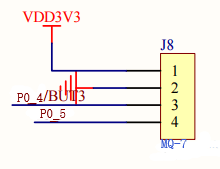


图5.10 MQ-7连线图

Fig.5.10 MQ-7 Connection diagram

### 5.3.4 AD转换设计

如图5.3所示，CC2530内置有12位的AD转换器，所以在本系统中，我们的设计可以不需要新增外部的AD转换，只需要根据厂商所提供的使用方式配置寄存器来完成AD转换，AD转换驱动代码与具体实现在第六章中6.2.1小节终端数据采集节点软件设计中进行介绍。

## 5.4 协调器节点硬件电路设计

协调器节点的硬件电路设计在本系统中目前只集成了一个功能，即为串口通信电路。所谓串口通信，是用来与PC端进行通讯的部分。在组网建立成功之后，接受到终端采集的数据需要经过协调器，依靠串口通信电路来与PC机进行数据传输。

串口通信电路的设计靠USB转串口来代替传统串口与串口之间的传输，驱动电路依靠芯片CH341来实现的，其驱动电路如图5.11所示。

数据从CC2530的各I/O口（如图5.11连接方式），然后通过USB转串口驱动来将数据发送至PC端，当PC机与设备连接时，PC机会自行进行USB转串口驱动的安装。

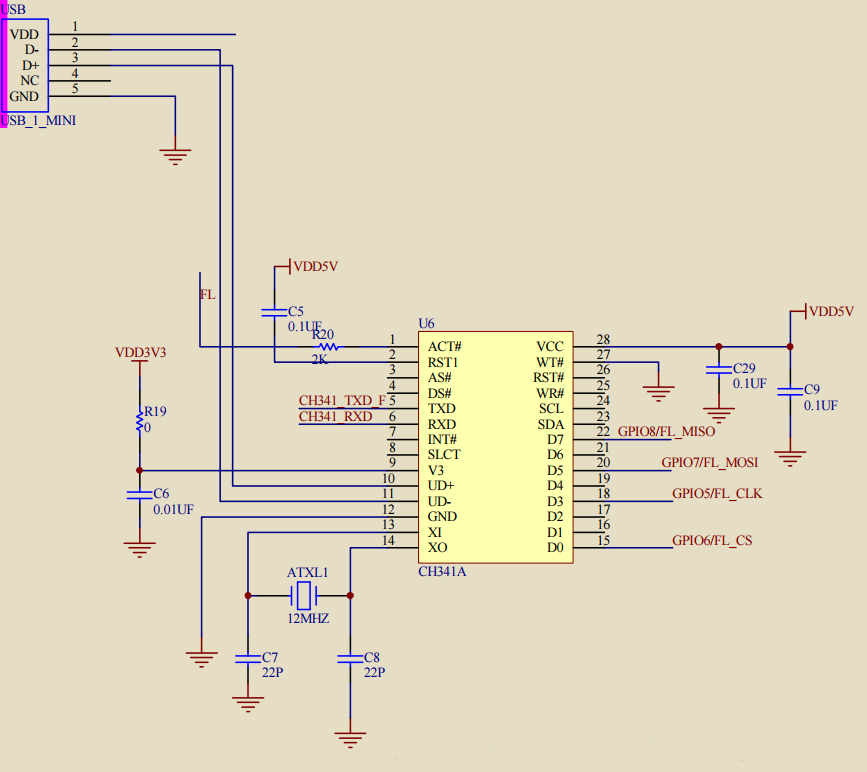


图5.11 USB驱动电路

Fig.5.11 USB drive circuit

## 5.5本章小结

本章在硬件总体框架上细化了硬件设计。在本文的设计中，硬件设计较为精简，完成的任务主要是从现场采集数据进行输出到PC端验证算法的可行性。

首先，设计终端节点硬件电路，主要包括传感器采集节点与AD转换过程，经过此步操作会将数据进行打包编码，准备转发。

在数据准备完毕，待协调器节点接收到数据后，通过串口将数据传送给PC，协调器硬件电路设计主要完成了USB转串口驱动电路的设计。

# 6 火灾报警系统的软件设计

本章是在火灾报警系统的软件总体架构的基础上进行细化设计。分别是对硬件的编程与PC监控软件的开发。

## 6.1 软件开发平台

### 6.1.1 IAR嵌入式软件开发平台

IAR嵌入式软件开发平台是由瑞典的IAR Systems公司专门为ARM嵌入式微处理器开发的集成开发环境。此开发软件能够方便用户对底层代码的开发。

IAR嵌入式开发软件上手速度快，简洁易懂。本系统主要是用IAR嵌入式软件开发平台实现对CC2530的功能开发。

### 6.1.2 ZStack协议栈开发平台

要是有ZigBee网络，需要有协议栈的支撑来实现系统功能。协议栈的工作模式为层与层通过APS接口发送或者接收操作请求。在协议栈中包含一个调度程序，即OSAL（操作系统抽象层），ZStack协议栈就是在OSAL上运行的[2]。

OSAL是一个具有多任务处理、事件轮询、消息处理机制的操作系统。当需要时操作时系统被唤醒，处理事件，其余情况下系统处于休眠状态。OSAL系统还具有消息队列处理机制，将事件按照紧急程度的执行顺序排列，然后从高到低执行事件操作。

如图6.1为项目中的ZStack结构目录。如图所示，在一个ZStack目录下，有一些是出厂时已经根据硬件设备设计好的，我们并不需要关心，拿本系统举例，例如MAC/NWK/OSAL等是厂家并不希望我们去更改的，我们在这里只需要关注的是App层与HAL层。App层主要是增加本系统的应用的具体程序，而HAL层是主要对硬件配置相关做具体的修改。

在系统启动时，会对系统所有的东西进行初始化，为后续网络建立做准备工作。然后系统跑起来后，就是OSAL的调度程序开始工作，实质上它是一个死循环，即轮询式查找状态，主要发现除PHY层以外所有层中发生的事件任务，事件的执行顺序为从下到上的的MAC层直到应用层。

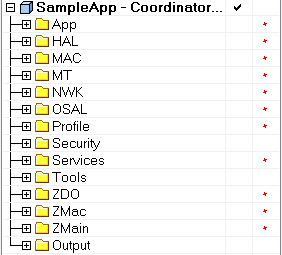


图6.1 系统软件总体架构

Fig.6.1 System Software Architecture

### 6.1.3 Microsoft Visual Studio软件开发平台

Microsoft Visual Studio（简称VS）是美国微软公司的开发工具包系列产品。

VS是一个基本完整的开发工具集，它包括了整个软件生命周期中所需要的大部分工具，如UML工具、代码管控工具、集成开发环境(IDE)等等。所写的目标代码适用于微软支持的所有平台，包括Microsoft Windows、Windows Mobile、Windows CE、.NET Framework、.NET Compact Framework和Microsoft Silverlight 及Windows Phone。

Visual Studio是目前最流行的Windows平台应用程序的集成开发环境。

## 6.2 节点软件设计

软件具体程序详见附录A，本小节只给出系统软件的流程图与框架。

### 6.2.1 终端数据采集节点软件设计

根据功能需求与节点采集软件的主要功能，该软件设计流程如下：

（1）软件在上电时开始初始化所有功能，然后循环扫描可用信道加入网络，等待加入网络成功响应。

（2）当终端节点成功加入组网后，就开始执行自己所负责的工作，即开始进行数据采集工作，将所采集的信号传递至路由器，再由路由器转发至协调器。

如图6.2所示为终端数据采集流程图。

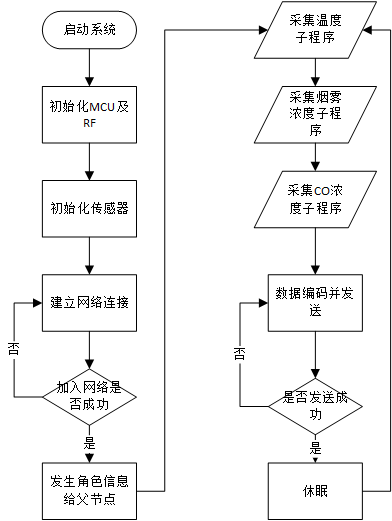


图6.2 终端数据采集流程图

Fig.6.2 Terminal data collection flow chart

### 6.2.2 路由节点的软件设计

路由器作为中间节点，主要维护的是将终端所采集的数据传送给协调器。如图6.3所示为路由节点流程图。

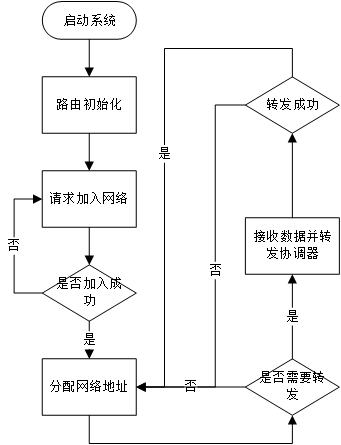


图6.3 路由节点软件流程

Fig.6.3 Routing node software flow

### 6.2.3 协调器节点的软件设计

协调器不仅是完成整个组网的建立与通信，还需要完成数据的收发。接受来自终端的数据且将接收到的数据通过串口转发给PC端。

协调器主要工作流程如下：

（1）系统上电后即初始化，然后进行网络的组网和建立，筛选出可用信道，接受设备的申请需求，分配网络地址。

（2）网络建立完成后，接受转发的数据信息，将数据通过串口发送给PC机。

（3）重复步骤（2）。

此时需要对协调器进行编程，由于ZigBee协议栈提供的应用层接口方便用户使用。本系统的传输方式采用点播通讯，在ZStack协议栈中已经提供相应的接口SampleApp\_Send\_P2P\_Message，我们只需要加入任务处理需求与应用程序代码即可。

协调器软件工作流程如图6.4所示。

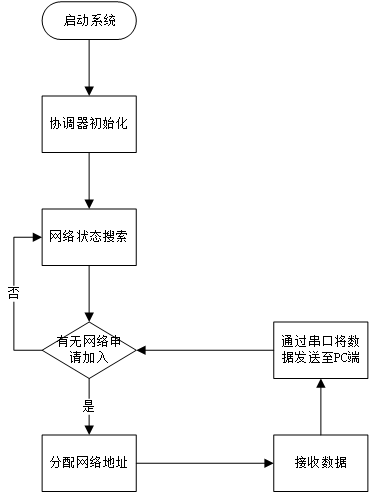


图6.4 协调器软件工作流程

Fig.6.4 Coordinator software workflow

当通讯类型为SAMPLEAPP\_P2P\_CLUSTERID时，通过HalUARTWrite(0, pkt->cmd.Data, pkt->cmd.DataLength)来完成数据传输给PC。

## 6.3 监控PC端软件设计

监控PC端软件主要是通过采集到串口传输的数据来进行数据融合，最终输出一个决策值。本系统软件设计思路考虑到多线程思想，为网络机器学习的过程专门开辟一个新线程，同时也为串口数据读取开辟一个新的线程，主线程执行其他扩展操作。整个代码的功能块主要划分为5个部分：串口通讯程序、阈值检测算法、遗传算法程序、BP神经网络算法程序、模糊逻辑算法程序。

由于考虑到论文的精简与突出重点，本章只介绍比较简单的代码块，比较复杂的代码块用流程图、伪代码的形式表现，具体每个模块代码详见附录B。

### 6.3.1 串口通讯程序

如图6.5所示为本系统串口设计类图。



图6.5 串口通信类图

Fig.6.5 Serial port communications class diagram

串口通讯程序是使用第三方开源库QT库中<QSerialPort>与<QSerialPortInfo>来完成的，该串口通讯程序主要是完成从协调器串口读取数据与解码工作，然后发送至神经网络线程中进行决策输出。

串口初始化流程与串口通讯程序流程分别如图6.6和6.7所示：

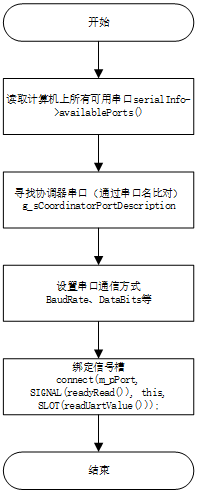
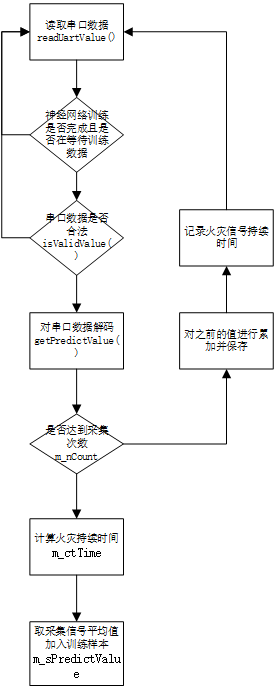
 

图6.6 串口初始化流程 图6.7 串口通讯程序流程

Fig.6.6 Serial initialize flow Fig.6.7 Serial port initialization process

### 6.3.2 阈值检测算法程序

此处的阈值检测算法为数据层融合器，仅仅只是过滤掉非火灾信号。

算法代码如下：

……

if (ALARM\_THRESHOLD < temp.inputValue.at(0) || ALARM\_THRESHOLD < temp.inputValue.at(1) || ALARM\_THRESHOLD < temp.inputValue.at(2))

{

……

}

if (ALARM\_THRESHOLD > temp.inputValue.at(0) && ALARM\_THRESHO\

LD > temp.inputValue.at(1) && ALARM\_THRESHOLD > temp.inputValue.at(2))

{

……

}

……

### 6.3.3 遗传算法程序

如图6.8所示为本系统遗传算法类的设计图。



图6.8 遗传算法类图

Fig.6.8 Genetic algorithm class diagram

由于遗传算法的流程图已在3.1.1节图3.1中介绍，此处不再介绍遗传算法的流程图。

针对本系统的遗传算法，设定初始化遗传代数GA\_GENERATIONS\_NUMB-ER为100000代；初始种群规模GA\_POPULATION\_SCALE为100代。

此时选出的最佳的个体的阈值和各层之间的权值作为BP神经网络算法的初始化阈值和权值。

在这里，特别说明遗传算子的编解码形式采用宏定义，以提高系统运算的效率。编解码的宏定义如下：

#define GA\_ENCODE\_STYLE(x) ((x) \* (GA\_RANGE - 1)) / (GA\_MAX\_WE\

IGHT - GA\_MIN\_WEIGHT)

#define GA\_DECODE\_STYLE(x) ((double)(x) \* (double)(GA\_MAX\_WEIGHT - GA\_MIN\_WEIGHT) / (double)(GA\_RANGE - 1))

### 6.3.4 BP神经网络算法程序

如图6.9所示为本系统BP神经网络类图。



图6.9 BP神经网络类图

Fig.6.9 BP neural network class diagram

BP神经网络算法流程图参照3.1.2节中图3.7中介绍。

在BP神经网络算法中，核心代码主要是网络训练代码块与网络预测代码块。

1．网络训练代码块。

void BPNeuralNetworks::training(samples learnValueList, double rateThreshold\

Value, double errorThresholdValue, bool bothMatch)

{

……

while (1)

{

if (bothMatch)

{

…… //退出条件

}

else

{

…… //退出条件

}

……

//Start training

for (int iter = 0; iter < learnValueCount; iter++)

{

setInput(learnValueList[iter].inputValue);

setOutput(learnValueList[iter].outputValue);

sendForwardPropagation(); //前馈计算

sendBackPropagation(); //反馈计算

}

function1(); //更新输入层权值与阈值

function2(); //更新隐藏层权值与阈值

function3(); //更新输出层阈值

……

}

}

2．网络预测代码块。

void BPNeuralNetworks::predict(samples& testValueList)

{

……

for (int it = 0; it < testValueCount; it++)

{

//前馈计算

}

}

特别说明，神经网络算法训练的退出条件为最终误差率（finalErrorRate）与最终误差（finalError）分别满足条件（即小于既定阈值）时，则完成训练。

对于finnalErrorRate与finalError，算法中所定规则finnalErrorRate与finalError的表达式如下所示：

 （6-1）

 （6-2）

 —— 输出层预测值；

 —— 输出层期望值；

i —— 输出层节点数；

m —— 神经网络训练数据组数。

### 6.3.5 模糊逻辑算法程序

图6.10为模糊逻辑算法的类图。



图6.10 模糊逻辑算法类图

Fig.6.10 Fuzzy logic algorithm class diagram

当接收到一组数据的时候会对数据的隶属度进行判断，然后再在所录入的表信息进行查找比对，最终输出决策值。

如图6.11流程图所示为模糊逻辑算法。

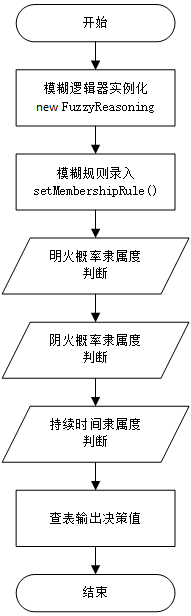


图6.11 模糊逻辑算法流程图

Fig.6.11 Fuzzy logic algorithm flow chart

## 6.4 软件整体工作流程

本部分介绍整个监测软件的工作流程，将所有模块组合起来了解会更清晰整个系统的工作模式。如图6.12所示为整个系统的流程。

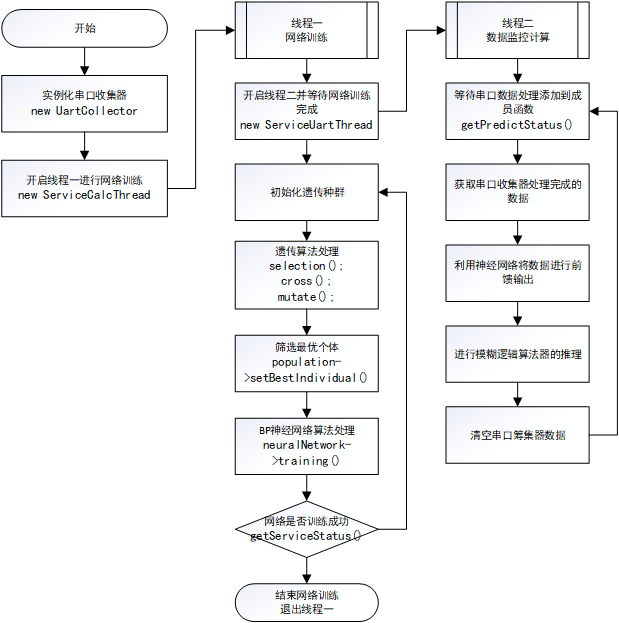


图6.12 软件整体工作流程

Fig.6.12 Software flow chart

## 6.5 本章小结

本章从两个部分介绍了整个系统的软件部分：一是节点软件的设计；二是监控PC端软件的设计。总的来说，节点软件主要是终端采集程序与协调器通信程序的编程，需要确定通信方式，本系统采用ZStack中已有的点对点通信模式，而监控软件的核心在于如何训练出神经网络，本节都对这些部分进行一个详细的介绍。

最后对系统做了一个总结，将软件工作流程的整体流程图进行梳理，使系统的工作流程更加清晰。

# 7 模拟与实验结果分析

## 7.1 实验环境搭建

验证数据融合算法的正确性，需要搭建硬件环境来采集数据现场，根据硬件电路设计方案，组件的终端节点与协调器节点设计采用模块化的方式，分别如图7.1与图7.2所示。

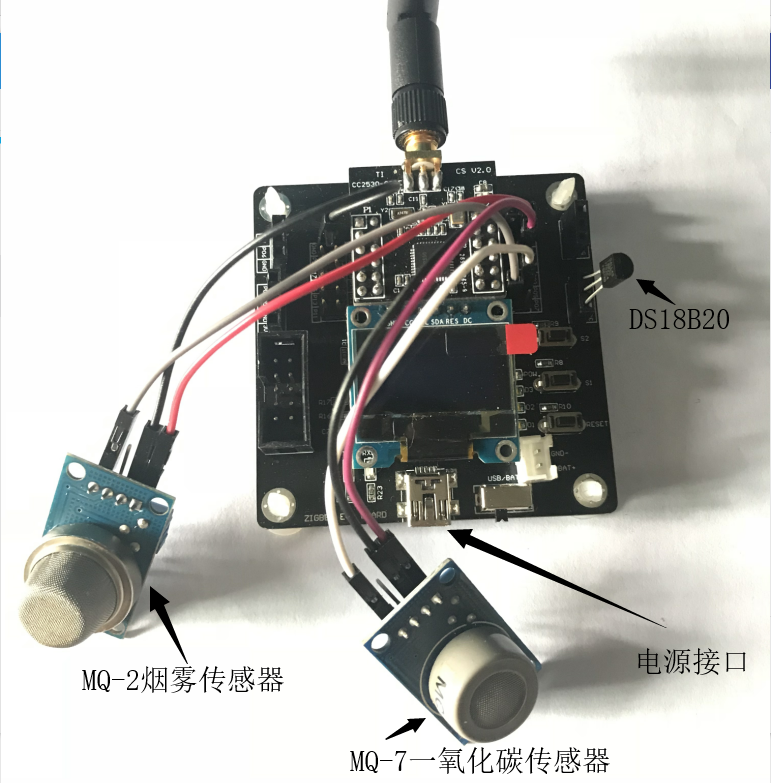
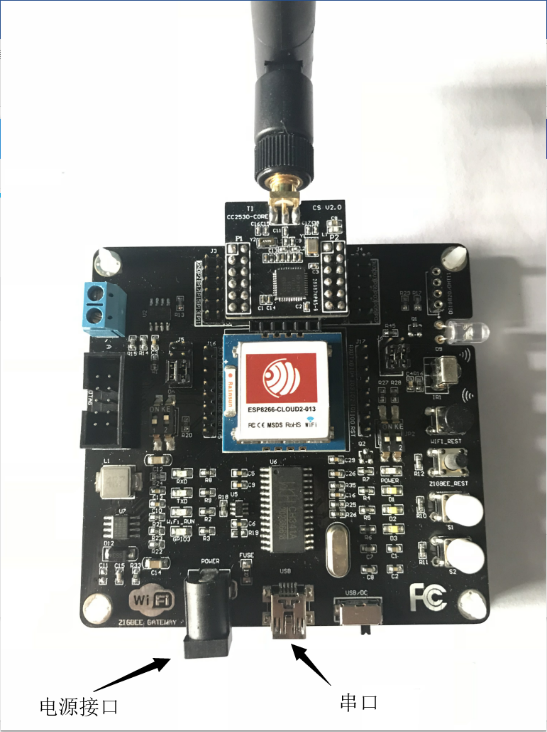
 

图7.1 终端节点实物图 图7.2 协调器节点实物图

Fig.7.1 Terminal node physical map Fig.7.2 Coordinator node physical map

将终端与协调器的软件提前烧录至芯片，在上电之后即可完成组网通信。特别注意的是，协调器节点的串口必须与PC端进行连接。

在准备好硬件电路后，需要对PC端监控软件进行启动。

软件启动时，工作流程如软件设计所介绍的，首先会进行遗传算法来筛选最优种群个体，软件体现如下图所示。

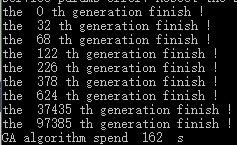


图7.3 监控软件遗传算法工作示意图

Fig7.3 Monitoring Software Genetic Algorithm Working Diagram

接着将种群中最优的个体作为神经网络算法的初始网络，随后将进行神经网络的训练，软件所呈现的内容如图7.4所示。

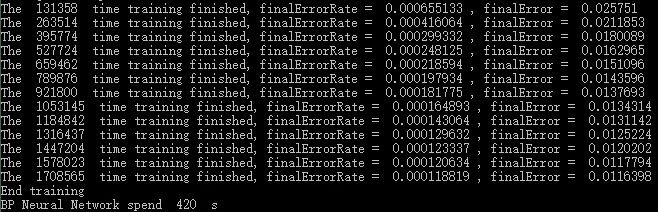


图7.3 监控软件神经网络训练工作示意图

Fig7.3 Monitoring software neural network training work diagram

针对本次模拟验证，finnalErrorRate与finalError的值分别设定为0.0001与0.01，当误差值达到上述要求之后，则神经网络准备完毕等待读取串口数据，软件体现如图7.4所示。



图7.4 监控软件成功启动

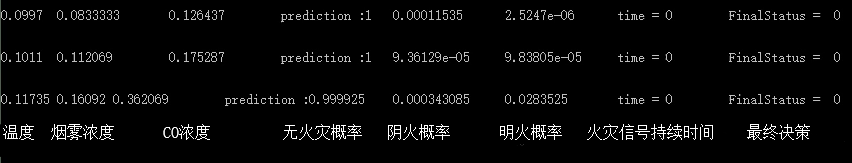
Fig7.4 Monitoring software successfully started

## 7.2 实验数据结果及分析

本次数据采集分四种模拟情形进行数据采集，分别如下：

1．无火灾发生。

当无火灾发生的时候，各传感器所采集参数及趋势图如图7.5和7.6所示。



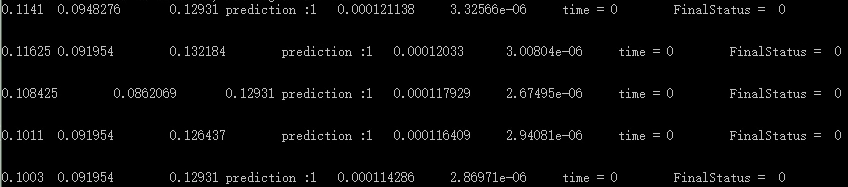


图7.5 无火灾情况所采集的数据

Fig7.5 No fire data collected

图7.6 无火灾情况趋势图

Fig7.6 No fire situation trend

**（注：这里对各参数代表的意思做相应的解释，特别说明的是，FinalStatus值0表示无火灾，1表示有火灾，后面的模拟情形同理。）**

如图所示，在无火灾发生时，各参数经过数据融合后的决策值输出为0表示无火灾发生，且预测率无火灾概率几乎等于0，表明系统精度可靠，验证无误。

2．模拟明火火灾。

当模拟明火情形的时候，各传感器所采集参数及趋势图如图7.7和7.8所示。

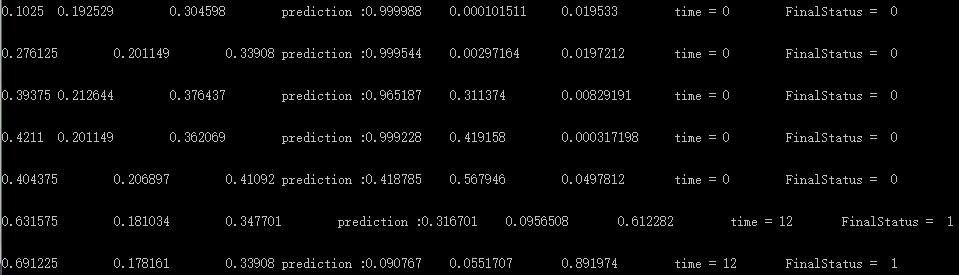


图7.7 明火火灾情况所采集的数据

Fig7.7 Data collected from open fire conditions

图7.8 明火火灾趋势图

Fig7.8 Fire fire trend chart

如图所示，模拟明火情形的时候，温度传感器与CO传感器的的上升速率较快，且根据预测概率，明火火灾概率急剧上升，且火灾持续信号超过预设阈值（10s），最终决策输出值为火灾发生，验证无误。

3．模拟阴火火灾。

当模拟阴火情形的时候，各传感器所采集参数及趋势图如图7.9和7.10所示。

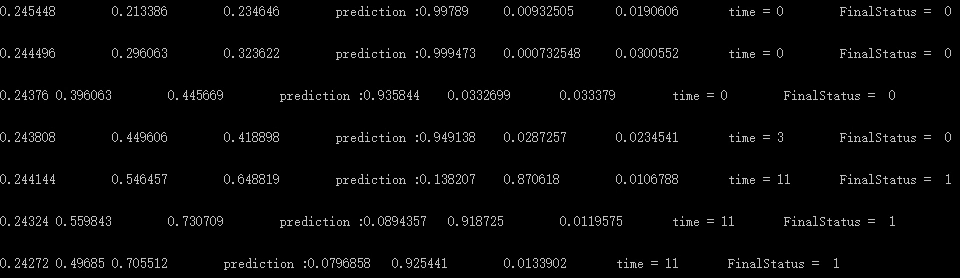


图7.9 阴火火灾情况所采集的数据

Fig7.9 Data collected during a fire in a fire

图7.10 阴火火灾趋势图

Fig7.10 Firefighting Trend Chart

如图所示，模拟阴火火灾情形的时候，温度传感器的值几乎不发生改变，相反的是，烟雾传感器与CO传感器的值上升速率较快，且根据预测概率，阴火火灾概率急剧上升，且火灾持续信号超过预设阈值（10s），最终决策输出值为火灾发生，验证无误。

4．恢复正常。

当火灾扑灭后，采集数据如下所示。

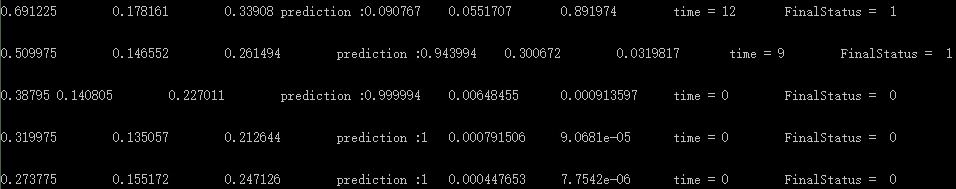


图7.11 扑灭火灾后采集数据

Fig7.11 Collect data after extinguishing fire

恢复正常后，各传感器所采集参数如图7.12所示。

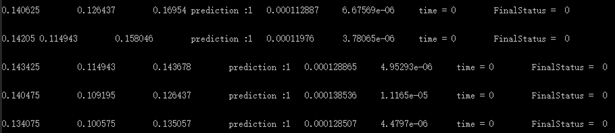


图7.12 恢复正常后采集数据

Fig7.12 Restore data after normal recovery

针对上述扑灭火灾后所采集的数据，绘制折线图所表示的趋势如图7.13所示。

图7.13 恢复正常后火灾趋势

Fig7.13 Restore data after normal recovery

如图所示，三类参数均恢复到与无火灾发生时的传感器所采集到的参数相近，且依然呈缓慢下降趋势。

## 7.3 本章小结

本章通过搭建硬件环境，模拟创建火灾情形，来验证系统的可行性。通过采集的数据进行数据融合处理，能达到的对阴火火灾与明火火灾统一报警。由数据分析表明，本系统能够准确监控三大参数的变化情况，且本系统给了火灾一个缓冲时机，通过加入危险信号持续时间来作为决策值，接受持续时间较长的采集信号，所以很少有中间概率产生的情况，排除了偶然的情况发生，更增加了系统的可靠性。

本系统的验证环境是在狭小的空间内，通信距离在5m左右完成的。实验证明了本系统的软件与硬件的可行性，

# 8 结论及展望

## 8.1 总结

本系统的设计抓住了传统设备的痛点，把无线传感器网络和ZigBee技术运用到了无线火灾报警系统中，并且利用数据融合算法来对传感器探测节点的数据进行融合。该系统既克服了传统设备的布线工序的复杂，也解决了因单传感器导致的误报与漏报情况的产生。本论文的具体完成内容如下。

1．分析了国内外火灾报警系统的研究现状。

2．将无线传感器网络与ZigBee技术运用到火灾报警系统中。

3．深入剖析了本系统的数据融合算法，然后针对于本系统来建立了基于遗传算法优化的BP神经网络算法的机器学习的过程，同时也确定了模糊逻辑的判断规则。

4．提出了系统总体方案设计，包括处理器与传感器选型，软硬件设计架构。采用CC2530作为系统的主控制器芯片，传感器节点采用DS18B20、MQ-2、MQ-7来监测现场参数。软件编程主要思想参照模块化，确定了软件各模块需要的设计内容。

5．完成了系统硬件的设计。包括终端节点电路与协调器节点电路的设计。

6．在ZStack协议栈的基础上设计了终端与协调器的软件编写，给出了各功能模块的流程图与伪代码。再利用VS，使用C++语言对监控软件进行了编写。

7．对系统进行了联调。模拟火灾情景，收集现场实验数据与结果。测试结果表明。系统运行可靠，达到预期目标。

## 8.2 展望

由于时间与个人水平有限。本文目前只完成了在实验阶段的测试，并没有能力实际运用到楼宇火灾的情况中，且系统的网络算法有时运行时间较长，训练数据规模较小，导致特殊情况的数据不精确，今后系统需要从以下几点去改进或者是升级系统。

1．数据融合。

对于神经网络算法来讲，需要提升该系统的训练数据规模，改进神经网络算法学习率的自适应。对于遗传算法需要寻找一个最优的适应度表达公式，寻找最优的变异或交叉的算法。

2．报警功能完善。

本系统暂时没有增添报警电路，后续升级可以根据系统输出的标识符（0或1）来驱动报警电路，实现应该比较简单。

3．上位机的制作

目前是通过字符界面来进行显示，其实对于实际生活的需求来讲，还需要制作直观的上位机方便观察和操作。

4．报警善后处理。

要真正做到智能化，可以根据火灾现场的火势大小（可以采用摄像头，利用图像算法来识别火势大小）采取不同的善后处理措施。

# 致谢

到此，本系统的设计工作与论文撰写工作也即将完成。这篇文章是对我大学四年生活的一个总结，也将我大学四年的所有知识运用在一起完成了这个系统的研究。

回首四年的生活，欢声笑语、酸甜苦辣的校园生活丰富了我的人生。在大学学到了很多课本上的知识，以及课本上学不到的一些待人处事的道理。这次论文的撰写是对我大学生活的一个总结，是对西南石油大学的留念。

首先，感谢西南石油大学四年的栽培，四年的收获，是我一辈子的财富，四年的时光是我人生中美丽的回忆。

其次，感谢我的指导老师——罗宾老师。从进入西南石油大学，所有的实验课都是罗老师一直在带我们，罗老师的授课方式我十分喜欢。这次罗老师带我毕业设计更增加了我对罗老师的感激之心，罗老师对我的毕设任务十分负责。一日为师终身为父，很珍惜这学期以来能和老师接触的时间这么多，感谢罗老师一直对我的指导关心和督促。

其次，感谢陪伴我大学四年的舍友、同学们。感谢你们给我大学四年的生活增添了很多色彩，大学因你们而精彩。

最后，感谢所有的答辩老师，谢谢你们能够耐心得对我论文进行评审，感谢老师们的悉心指教，感谢所有老师复出的辛勤劳动。

# 参考文献

1. 力安科技. 2017年1至10月全国火灾事故数据统计：共接报火灾21.9万起[EB/OL]. http://www.gdliontech.cn/content/?1144.html, 2017-11-8
2. 李吉宗. 基于ZigBee的智能楼宇火灾报警系统研究[D]. 上海:华东理工大学, 2014. 5-7
3. 马祺. 移动自组织网络中的角度路由协议[J]. 北京:《外军电信动态》, 2004
4. 颜学义. 基于ZigBee的智能火灾报警系统设计[D]. 《国防科学技术大学》, 2008. 6-10
5. 李铁鹰, 秦茂盛. 基于ZigBee技术的家用安防系统设计[J]. 《仪器仪表用户》, 2011, 18(4): 27-29
6. 丁雪莲. ZigBee协议栈浅析[J]. 《电脑与信息技术》, 2013, 21(5): 18-21
7. 白云龙. 基于ZigBee的无线消防报警系统设计[D]. 辽宁:辽宁工业大学, 2016. 5-7
8. 陈霄. DNA遗传算法及应用研究[D]. 浙江:《浙江大学》, 2010. 2-4
9. 满丽欣. SDN网络下基于BP神经网络算法的负载均衡研究[D]. 吉林:《吉林大学》, 2017. 21-26
10. 赵利民. 一种改进的BP神经网络算法及其在集中供热系统中的应用[D]. 吉林:《吉林大学》, 2012. 5-11
11. 赵炎. 信息融合技术在图书馆情报系统中的应用分析[J]. 《科技创新与应用》, 2014, (19): 299-299
12. 周创创. 基于无线传感网络的楼宇火灾监测系统研究[D]. 西安:《西安建筑科技大学》, 2016. 15-22

# 附录A 节点软件设计

1．温度采集程序

float floatReadDs18B20(void)

{

unsigned char V1,V2;

unsigned int temp;

float fValue;

Ds18b20Initial();

Ds18b20Write(0xcc);

Ds18b20Write(0x44);

Ds18b20Initial();

Ds18b20Write(0xcc);

Ds18b20Write(0xbe);

V1 = Ds18b20Read();

V2 = Ds18b20Read();

temp=V2\*0xFF+V1;

fValue = temp\*0.0625;

return fValue;

}

2．烟雾浓度采集及AD转换程序

uint16 readCGData(void)

{

uint16 reading = 0;

ADCCFG |= 0x40;

ADCCON3 = 0x86;

while (!(ADCCON1 & 0x80));

ADCCFG &= (0x40 ^ 0xFF);

reading = ADCL;

reading |= (int16)(ADCH << 8);

reading >>= 8;

return (reading);

}

3．CO浓度采集及AD转换程序

uint16 readCOData(void)

{

uint16 reading = 0;

ADCCFG |= 0x40;

ADCCON3 = 0x85;

while (!(ADCCON1 & 0x80));

ADCCFG &= (0x40 ^ 0xFF);

reading = ADCL;

reading |= (int16)(ADCH << 8);

reading >>= 8;

return (reading);

}

4．协调器通讯程序

void SampleApp\_Send\_P2P\_Message( void )

{

char precdictValue[30]; //待传输数据

uint16 CGData;

uint16 COData;

float temperature;

temperature = floatReadDs18B20(); //DS18B20数据子模块

CGData = readCGData(); //烟雾浓度读取子模块

COData = readCOData(); //CO浓度读取子模块

sprintf(precdictValue, "TM%.02f|CG%d|CO%d", temperature, CGData, COData); //数据编码，方便PC端程序使用

if ( AF\_DataRequest( &SampleApp\_P2P\_DstAddr, &SampleApp\_epDesc,

SAMPLEAPP\_P2P\_CLUSTERID,

strlen(precdictValue),

precdictValue,

&SampleApp\_TransID,

AF\_DISCV\_ROUTE,

AF\_DEFAULT\_RADIUS ) == afStatus\_SUCCESS )

{

} //数据发送

}

# 附录B 监控软件设计

此处先会对固定的串口进行一个全局定义，便于终端与协调器与PC相连时进行串口的查询与配置。

static const QString g\_sCoordinatorPortDescription = "USB-SERIAL CH341A";

static const QString g\_sTerminalPortDescription = "USB Serial Port"

1．串口初始化程序核心代码

foreach(QSerialPortInfo winPort, serialList)

{

if (g\_sTerminalPortDescription == winPort.description() || g\_sCoordinator\

PortDescription == winPort.description())

{

m\_sSerialDescription = winPort.description();

m\_sSerialName = winPort.portName();

}

}

if (!m\_sSerialDescription.isNull() && !m\_sSerialName.isNull())

{

m\_pPort = new QSerialPort(m\_sSerialName);

m\_pPort->setBaudRate(QSerialPort::Baud115200);

m\_pPort->setDataBits(QSerialPort::Data8);

m\_pPort->setParity(QSerialPort::NoParity);

m\_pPort->setStopBits(QSerialPort::OneStop);

m\_pPort->setFlowControl(QSerialPort::NoFlowControl);

if (!m\_pPort->open(QIODevice::ReadOnly))

{

qDebug() << "Open Failed !";

}

QObject::connect(m\_pPort, SIGNAL(readyRead()), this, SLOT

(readUartValue()));

}

2．串口数据读取核心代码

void UartCollector::readUartValue()

{

if (!m\_bIsReadyPredict && m\_bServiceRun)

{

QString portValue(m\_pPort->readAll());

if (isValidValue(portValue))

{

if (UART\_OBTAIN\_COUNT > m\_nCount)

{

sample temp = getPredictValue(portValue);

if (!m\_bHasStartTime)

{

if (ALARM\_THRESHOLD < temp.inputValue.at(0) || \

ALARM\_THRESHOLD < temp.inputValue.at(1) ||\

ALARM\_THRESHOLD < temp.inputValue.at(2))

{

m\_ctStart = clock();

m\_bHasStartTime = true;

}

}

if (ALARM\_THRESHOLD > temp.inputValue.at(0) &&\

ALARM\_THRESHOLD > temp.inputValue.at(1) && \

ALARM\_THRESHOLD > temp.inputValue.at(2))

{

m\_ctEnd = clock();

m\_bHasEndTime = true;

}

if (0 == m\_sPredictValue.inputValue.size())

{

m\_sPredictValue.inputValue.push\_back(temp.inputValue[\

0]);

m\_sPredictValue.inputValue.push\_back(temp.inputValue[\

1]);

m\_sPredictValue.inputValue.push\_back(temp.inputValue[\

2]);

}

else

{

m\_sPredictValue.inputValue[0] = m\_sPredictValue.input\

Value.at(0) + temp.inputValue.at(0);

m\_sPredictValue.inputValue[1] = m\_sPredictValue.input\

Value.at(1) + temp.inputValue.at(1);

m\_sPredictValue.inputValue[2] = m\_sPredictValue.input\

Value.at(2) + temp.inputValue.at(2);

}

}

else

{

if (!m\_bHasStartTime)

{

m\_ctStart = clock();

}

if (!m\_bHasEndTime || m\_ctEnd < m\_ctStart)

{

m\_ctEnd = clock();

}

QMutexLocker locker(m\_pMutex);

m\_sPredictValue.inputValue[0] = m\_sPredictValue.input\

Value.at(0) / UART\_OBTAIN\_COUNT;

m\_sPredictValue.inputValue[1] = m\_sPredictValue.input\

Value.at(1) / UART\_OBTAIN\_COUNT;

m\_sPredictValue.inputValue[2] = m\_sPredictValue.input\

Value.at(2) / UART\_OBTAIN\_COUNT;

m\_oPredictValueList.push\_back(m\_sPredictValue);

m\_ctTime = (m\_ctEnd - m\_ctStart) / CLOCKS\_PER\_SEC;

m\_bIsReadyPredict = true;

m\_sPredictValue.inputValue.clear();

m\_sPredictValue.outputValue.clear();

}

m\_nCount++;

}

}

}

3．模糊逻辑核心代码

double FuzzyReasoning::normalMembership(const double value)

{

double PSMembership = gaussian(value, FUZZY\_PS);

double PMMembership = gaussian(value, FUZZY\_PM);

double PBMembership = gaussian(value, FUZZY\_PB);

/\* Get the max menbership, such as PS, PM, PB \*/

if (PSMembership > PMMembership)

{

if (PSMembership > PBMembership)

{

return FUZZY\_PS;

}

else

{

return FUZZY\_PM;

}

}

else if (PMMembership > PBMembership)

{

return FUZZY\_PM;

}

else

{

return FUZZY\_PB;

}

}

void FuzzyReasoning::setMembershipRule(const double fuzzyRuleList[]\

[FUZZY\_RULE\_COLUMN])

{

for (int i = 0; i < FUZZY\_RULE\_ROW; i++)

{

FuzzyRule fuzzySingleRule;

fuzzySingleRule.isFireRate = fuzzyRuleList[i][0];

fuzzySingleRule.likeFireRate = fuzzyRuleList[i][1];

fuzzySingleRule.time = fuzzyRuleList[i][2];

fuzzySingleRule.finalStatus = fuzzyRuleList[i][3];

m\_oFuzzyRules.push\_back(fuzzySingleRule);

}

}

4．网络训练线程核心代码

void ServiceCalcThread::run()

{

clock\_t printStartTime = clock();

ServiceUartThread \*uartThread = new ServiceUartThread;

uartThread->start();

while (1)

{

clock\_t startGATime = clock();

GAPopulation \*population = new GAPopulation(m\_oLearnValueList);

for (int i = 0; i < GA\_GENERATIONS\_NUMBER; i++)

{

/\* Genetic algorithm \*/

population->selection();

population->cross();

population->mutate();

clock\_t printEndTime = clock();

if (PRINT\_TIME\_TEN < (printEndTime - printStartTime) / clock\_t(2))

{

printStartTime = printEndTime;

qDebug() << "the " << i << "th generation finish !";

}

}

population->setBestIndividual();

clock\_t endGATime = clock();

qDebug() << "GA algorithm spend " << (endGATime - startGATime) / CLOCKS\_PER\_SEC << " s";

clock\_t startBPTime = clock();

BPNeuralNetworks \*neuralNetwork = population->getBestIndividual();

neuralNetwork->training(m\_oLearnValueList, BP\_RATE\_THRESHOLD , BP\_ERROR\_THRESHOLD, true);

clock\_t endBPTime = clock();

qDebug() << "BP Neural Network spend " << (endBPTime - startBPTime) / CLOCKS\_PER\_SEC << " s";

/\* Get server status, restart if not successful \*/

if (!neuralNetwork->getServiceStatus())

{

qDebug() << "Service params error! Reboot the service now!";

population = nullptr;

}

else

{

/\* Service begin, set the collector & BP network \*/

uartThread->setCollector(m\_pCollector);

uartThread->setBPNeuralNetworks(neuralNetwork);

break;

}

}

}

5．串口数据预测线程核心代码

void ServiceUartThread::run()

{

qDebug() << "Waiting for service begin!";

while(1)

{

if (m\_pBPNeuralNetworks)

{

if (!m\_bIsServiceRun)

{

m\_bIsServiceRun = true;

/\* Set the mutex & bool value for the serial port collector \*/

m\_pCollector->setMutex(m\_pMutex);

m\_pCollector->setServiceStatus(m\_bIsServiceRun);

qDebug() << "Congratulation, service is running!";

}

if (m\_pCollector->getPredictStatus())

{

QMutexLocker locker(m\_pMutex);

samples predictValueList = m\_pCollector->getPredict\

ValueList();

m\_pBPNeuralNetworks->predict(predictValueList);

for (int i = 0; i < predictValueList.size(); i++)

{

for (int j = 0; j < predictValueList[i].inputValue.size();\

j++)

{

cout << predictValueList[i].inputValue[j] << "\t";

}

cout << "prediction :";

for (int j = 0; j < predictValueList[i].outputValue.size();\

j++)

{

cout << predictValueList[i].outputValue[j] << "\t";

}

double timeRate = m\_pCollector->getTime() / 10.0;

if (timeRate > 1.0)

{

timeRate = 1.0;

}

cout << "time = " << m\_pCollector->getTime() << "\t";

FuzzyRule fuzzySingleRule;

FuzzyReasoning \*fuzzyReasoning = new \

FuzzyReasoning;

fuzzySingleRule.isFireRate = fuzzyReasoning->normal\

Membership(predictValueList[i].outputValue[2]);

fuzzySingleRule.likeFireRate = fuzzyReasoning->nor\

malMembership(predictValueList[i].outputValue[1]);

fuzzySingleRule.time = fuzzyReasoning->normal\

Membership(timeRate);

if (fuzzySingleRule.time == FUZZY\_PM)

{

fuzzySingleRule.time = FUZZY\_PB;

}

fuzzyReasoning->finalDecision(fuzzySingleRule);

qDebug() << "FinalStatus = " << fuzzySingle\

Rule.finalStatus;

cout << endl;

cout << endl;

delete fuzzyReasoning;

fuzzyReasoning = nullptr;

}

m\_pCollector->clearPredictValueList();

m\_pCollector->setPredictStatus(false);

m\_pCollector->setEndTimeStatus(false);

m\_pCollector->setStartTimeStatus(false);

m\_pCollector->setCount(0);

}

}

}

}