



工程硕士专业学位论文

基于STM32的智能控制器的研究与开发
Research and Development of Intelligent
Controller Based on STM32

作 者：邓凯旋
校内导师：翟延忠教授
企业导师：李红军高工

华北科技学院
2022年6月



工程硕士专业学位论文

基于STM32的智能控制器的研究与开发
Research and Development of Intelligent
Controller Based on STM32

作 者：邓凯旋

校内导师：翟延忠教授

企业导师：李红军高工

导师组组长：黎 冠

导师组成员：张全柱、翟延忠、彭程、
邓永红、于臻、张涛、靳文涛、苗志全、王江华、郭海文、叶
瑜、朱小龙、魏景新、尤文强、刘永涛

华北科技学院

2022年6月

学位论文使用授权声明

本人完全了解华北科技学院有关保留、使用学位论文的规定，同意本人所撰写的学位论文的使用授权按照学校的管理规定处理，即：①研究生在校攻读学位期间论文工作的知识产权单位属于华北科技学院，学校有权保存并向国家有关部门或机构送交论文的复印件和电子版，允许学位论文被查阅（除在保密期内的保密论文外）；②学校可以公布学位论文的全部或部分内容，可以使用影印、缩印或扫描等复制手段保存和汇编学位论文，学校可以将公开的学位论文作为资料在档案馆、图书馆等场所或在校园网上供校内师生阅读、浏览，学校可以将公布的学位论文编入有关数据库进行检索。另外，根据有关法规，同意中国国家图书馆保存研究生学位论文。

（保密的学位论文在解密后适用本授权书）。

本学位论文属于：

☐ 保密，在 年解密后适用本授权书。

☐ 不保密，同意在校园网上发布，供校内师生和与学校有共享协议的单位浏览。

（请在以上相应方框内打“√”）

作者签名：

校内导师签名：

年 月 日

企业导师签名：

年 月 日

中图分类号 TP273

学校代码 11104

UDC

密 级 公开

华北科技学院

工程硕士专业学位论文

基于STM32的智能控制器的研究与开发

Research and Development of Intelligent Controller
Based on STM32

作 者 邓凯旋

校内导师 翟延忠

企业导师 李红军

申请学位 工程硕士专业学位

培养单位 研究生部

学科专业 安全工程

研究方向 安全生产自动化和信息化

答辩委员会主席 洪利

评 阅 人

二〇二二年六月

论文审阅认定书

研究生邓凯旋在规定的学习年限内，按照研究生培养方案的要求，完成了研究生课程的学习，成绩合格；在我的指导下完成本学位论文，经审阅，论文中的观点、数据、表述和结构为我所认同，论文撰写格式符合学校的相关规定，同意将本论文作为学位申请论文送专家评审。

校内导师签字：

企业导师签字：

年 月

致谢

天可补，海可填，日月既往，不可复追。我即将走完研究生阶段，进入社会。回首往事如风，早已物是人非，唯有回忆，依旧如昨。三年以来，在我前进的道路上，有老师、家人、同学、朋友们的帮助，在综合实力上取得了显著进步。

首先非常感谢我的研究生导师翟延忠老师。翟老师把我录取进了实验室，提供了优秀的环境和教学资源，他知识渊博、见多识广给我的研究课题指引着前进的方向，同时循循善诱、尊重学生的想法，带领我在科研的领域中披荆斩棘。同时老师身上有很多优秀的品质，他为人谦和、公正严明，使我如沐春风。他处事有道、坚强上进，催促着我时刻修炼自身。有此良师，幸甚！

感谢实验室的师兄师姐欧阳昇、任凯、翟宝荣、马强，在我刚入门时对我的指导和帮助，提升我的工作效率，使我备受温暖。感谢同伴李燕帮我承担了实验室的工作，对我的帮助和照顾。感谢师弟白柯祯、周玉博对我的帮助关心。大家积极上进、和衷共济，营造了良好的实验室氛围，衷心祝愿你们在未来的道路上蓬勃发展、前程似锦。

感谢我的室友和挚友们，感谢你们的陪伴和鼓励！

感谢我的父母多年来的养育之恩，你们是我奋斗的动力。当我遇到难题时，你们总是鼓励我，帮我分担忧愁。

最后感谢所有参与论文评审的专家和答辩委员会的老师，衷心感谢你们的宝贵意见！

摘 要

工业生产过程控制中，经常有一些复杂任务的控制问题。企业一般采用 PLC 技术完成各个设备的监测、精确控制，从而提高效率。但是小型 PLC 使用传统 PID 算法面对非线性的系统、模型不确定的系统时难以完成复杂的过程控制任务，而基于 STM32 平台的智能控制算法在这方面比较擅长，可使整个系统具有不错的自动化、智能化的优点。

智能控制利用现代智能控制理论进行定性和定量分析，然后进行自动推理和决策，整个过程类似人脑的思维方式。本文研究了现代智能控制算法包括专家控制、神经网络控制、遗传算法、模糊控制、模型预测控制等的原理、特点和控制方法，实际以煤厂重介选煤工艺中水箱液位控制为应用案例进行研究与开发。具体过程是首先对水箱进行建模，明确整个采集和输出过程，而液位控制过程中的算法使用到了两种智能控制算法。一种是在经典 PID 基础上引入模糊控制器根据模糊规则进行模糊推理，实现自适应调节 PID 参数；另一种是运用现代控制理论的状态反馈控制算法对模型建立状态方程，得到反馈增益矩阵进行优化控制。

此外，为了更好的支持控制算法，采用了 STM32 为开发平台。鉴于其重量轻、体积小、编程功能强大、响应速度快、灵活性强等优点，在确保其高稳定性及较强抗干扰能力的前提下可替代小型 PLC 在工业控制现场的使用，其批量使用情形下均摊费用低廉，从经济效益考虑也利于推广使用，是一款不错的硬件开发平台。

综上设计了基于 STM32F407 的智能控制器，可支持多种智能控制算法。该控制器在最小系统基础上配有开关量和模拟量输入输出模块、电源模块、LCD 显示模块、存储模块，为其软件开发提供了基础硬件平台。软件开发架构上采用 FreeRTOS 系统管理多个任务。除了基本的控制开关、模拟量功能外还配套了其他的功能，如 QT 串口通信、数据存储、人脸识别、远程监控等等。

通过 MATLAB 仿真实验、在 STM32 控制器上的实现表明智能控制算法可获得预期的优化控制的效果，实现了给出目标液位智能控制器就会完成任务并且响应曲线符合快速性、准确性、稳定性的要求。同时该控制器也具备其他业务功能，整体上为基于 STM32 的智能控制器课题领域提供了一整套解决方案。

该论文有图 30 幅，表 3 个，参考文献 78 篇。

关键词：智能控制；模糊控制；PID；STM32

Abstract

In industrial production process control, there are often some complex task control problems. Enterprises generally use PLC technology to complete the monitoring of each equipment, accurate control, so as to improve efficiency. But the small PLC uses the traditional PID algorithm to face the nonlinear system, the model is uncertain when the system is difficult to complete the complex process control task, and the intelligent control algorithm based on STM32 platform is good at this aspect, can make the whole system has the advantage of good automation, intelligence.

Intelligent control uses modern intelligent control theory for qualitative and quantitative analysis, and then carries out automatic reasoning and decision-making, the whole process is similar to the way of thinking of human brain. This paper studies the principles, characteristics and control methods of modern intelligent control algorithms including expert control, neural network control, genetic algorithm, fuzzy control, model predictive control and so on. In this paper, the water tank level control in the heavy medium coal preparation process is studied and developed as an application case. First of all, modeling the water tank, clear the whole acquisition and output process, liquid level control process using two intelligent control algorithm. One is to introduce the fuzzy controller based on the classical PID to carry on the fuzzy reasoning according to the fuzzy rules and realize the adaptive adjustment of PID parameters. The other is to use the state feedback control algorithm of modern control theory to establish the state equation of the model and obtain the feedback gain matrix to optimize the control.

In addition, in order to better support the control algorithm, STM32 is adopted as the development platform. Given its light weight, small size, powerful programming, response speed, strong flexibility and etc, to ensure its high stability and strong anti-interference ability under the premise of the use of alternative small PLC in industrial control field, its bulk use case capitation fee is low, is conducive to promote the use of considering from economic benefits, is a good platform for the hardware development.

In conclusion, an intelligent controller based on STM32F407 is designed to support a variety of intelligent control algorithms. Based on the minimum system, the controller is equipped with switching and analog input and output modules, power modules, LCD display modules and storage modules, which provide the basic hardware platform for its software development. FreeRTOS system is used to manage multiple tasks in software development architecture. In addition to the basic control

switch, analog function but also supporting other functions, such as QT serial communication, data storage, face recognition, remote monitoring and so on.

Through MATLAB simulation experiments, the realization on STM32 controller shows that the intelligent control algorithm can obtain the effect of optimal control, and the intelligent controller will complete the task when the target liquid level is given, and the response curve meets the requirements of rapidity, accuracy and stability. At the same time, the controller also has other business functions, as a whole for the STM32 based intelligent controller subject field to provide a set of solutions.

There are 30 figures, 3 tables and 78 references in this paper.

Key words: intelligent control; Fuzzy control; PID; STM32

目 录

摘 要	I
目 录	IV
图清单	VIII
表清单	X
1 绪论	1
1.1 课题的背景及意义	1
1.2 课题研究的国内外发展现状	3
1.3 课题研究的主要内容和结构安排	7
2 智能控制器的需求分析	10
2.1 控制器的核心需求分析	10
2.2 其他需求分析	12
2.3 本章小结	12
3 控制算法的分析及选型	13
3.1 控制算法的分析	13
3.2 控制算法的选型	16
3.3 本章小结	16
4 智能控制器的硬件设计	17
4.1 硬件部分总体设计	17
4.2 硬件部分各模块设计	17
4.3 本章小结	22
5 智能控制器的软件设计	23
5.1 软件部分总体设计	23
5.2 智能控制器各功能模块的设计与实现	24
5.3 本章小结	45
6 智能控制器测试实验	46
6.1 智能控制器软件测试	46
6.2 本章小结	52
7 总结与展望	53
7.1 回顾与总结	53
7.2 主要技术成果	54

7.3 改进与展望	54
参考文献	58
作者简介	70
论文原创性声明	71
学位论文数据集	72

Contents

Abstract.....	I
Contents	IV
List of Figures	VIII
List of Tables	X
1 Introduction	1
1.1 Research background and significance	1
1.2 Research status at home and abroad	3
1.3 The main research content and structure of this paper	7
2 Demand analysis and of intelligent controller	10
2.1 Core requirement analysis of controller	10
2.2 Other Requirements Analysis	12
2.3 Chapter Summary	12
3 Analysis and selection of control algorithm	13
3.1 Analysis of control algorithm	13
3.2 Selection of control algorithm	16
3.3 Chapter Summary	16
4 Hardware design of intelligent controller	17
4.1 Design of hardware	17
4.2 Design of each module of the hardware part	17
4.3 Chapter Summary	22
5 Software design of intelligent controller	23
5.1 Design of software	23
5.2 Design and implementation of each power module of intelligent controller	24
5.3 Chapter Summary	45
6 Test of intelligent controller	46
6.1 Software test of intelligent controller	46
6.2 Chapter Summary	52
7 Summary and Prospect	53
7.1 Review and summary	53
7.2 Main technical achievements	54
7.3 The improvement and prospect	54

References	58
Author's Resume	70
Declaration of Thesis Originality	71
Thesis/Dissertation Data Collection	72

图清单

图序号	图名称	页码
图 1-1	技术路线图	8
Figure 1-1	Technical route diagram	8
图 3-1	专家整定 PID控制系统结构图	13
Figure 3-1	Diagram of Expert tuning PID control system structure	13
图 3-2	NN网络结构	14
Figure 3-2	Diagram of NN network structure	14
图 3-3	基于 BP 网络的 PID控制系统	15
Figure 3-3	Diagram of PID control system based on BP network	15
图 4-1	控制器硬件总体设计	17
Figure 4-1	Design of controller hardware	17
图 4-2	3.3V 电源原理图	18
Figure 4-2	3.3V power diagram	18
图 4-3	存储模块原理图	18
Figure 4-3	diagram of storage modules	18
图 4-4	LCD原理图	19
Figure 4-4	Diagram of LCD	19
图 4-5	模拟量输入原理图	19
Figure 4-5	schematic diagram of Analog input	19
图 4-6	CAN 模块原理图	20
Figure 4-6	Diagram of the CAN bus module	20
图 4-7	PWM输出模块原理图	20
Figure 4-7	Diagram of PWM output module	20
图 4-8	光耦输入模块原理图	21
Figure 4-8	diagram of Optocoupler input module	21
图 4-9	继电器输出模块原理图	21
Figure 4-9	diagram of Relay output module	21
图 4-10	控制器实物图	22
Figure 4-10	Physical picture of intelligent controller	22
图 5-1	PWM工作原理图	26
Figure 5-1	Diagram of PWM working principle	26
图 5-2	反馈控制系统结构图	27
Figure 5-2	Diagram of Feedback control system structure	27
图 5-3	模糊自适应PID框图	29
Figure 5-3	Diagram of Fuzzy adaptive PID block	29
图 5-4	模糊PID输入输出图	29
Figure 5-4	Diagram of Fuzzy PID input and output	29
图 5-5	模糊规则图	30
Figure 5-5	Diagram of Fuzzy rule	30

图 5-6	经典PID和模糊PID图	32
Figure 5-6	Diagram of Classic PID and Fuzzy PID	32
图 5-7	串口通信界面	34
Figure 5-7	Diagram of serial port communication	34
图 5-8	汉字取模图	38
Figure 5-8	Diagram of Chinese character generation font	38
图 5-9	代码上传到云端图	45
Figure 5-9	Diagram of the code is uploaded to the cloud	45
图 6-1	无专家误差响应曲线	46
Figure 6-1	Diagram of no expert error response curve	46
图 6-2	专家响应曲线和误差响应曲线	47
Figure 6-2	Expert response curve and error response curve	47
图 6-3	神经网络PID阶跃响应曲线	47
Figure 6-3	Neural network PID step response curve	47
图 6-4	经典PID和模糊PID仿真结果对比图	48
Figure 6-4	Comparison diagram of classical PID and fuzzy PID simulation results	48
图 6-5	模糊PID控制LCD响应曲线	48
Figure 6-5	Diagram of Fuzzy PID control LCD response curve	48
图 6-6	状态反馈控制LCD响应曲线	49
Figure 6-6	Diagram of State feedback control LCD response curve	49
图 6-6	死锁检测效果图	52
Figure 6-7	Diagram of deadlock detection	52

表清单

表序号	表名称	页码
表2-1	传统和现代控制比对	10
Table 2-1	Comparison of traditional and modern control	10
表3-1	智能控制算法汇总	16
Table 3-1	Intelligent control algorithm summary	16
表6-1	模糊PID和状态反馈汇总	49
Table 6-1	Fuzzy PID and state feedback summary	49

1 绪论

1 Introduction

1.1 课题的背景及意义 (Research background and significance)

1.1.1 课题提出的背景

智能控制器的提出来自企业重介选煤工艺过程。为了响应节能减排、精简能源结构,严格控制精煤灰分是煤炭生产企业的重中之重。企业生产中采取的重介选煤作为分选手段之一,关键是保障悬浮液液位和密度等相关变量的智能控制,进而提升自动化和智能化水平。重介选煤的基本原理是:将混有矸石的原煤放在合介桶中,加入水和某种介质使混合液密度增加,这样精煤密度小、浮在上层,矸石下沉槽的底部,利用阿基米德原理分离开来^[1-3]。因此控制好介质桶内液体的液位、密度变量很关键^[4]。企业具体的生产流程,发现主要的需求就是实时获取密度和液位的信号并控制分流阀和加水阀门的开度,控制算法采用传统的 PID 算法,但是不够智能化,参数不能实时调节、控制任务不能复杂化,从而一定程度上影响了控制的效果。

不仅控制算法需完善,硬件平台也有改进空间。企业往往采购成套的小型 PLC 系统去实现智能控制各个生产模块,但是采用外商的封装好的小型 PLC 装置,价格昂贵不说,PLC 梯形图编程逻辑较为简单,不可以用作专用控制器,难以胜任复杂度高、要求高的任务。总体上,功能不单一、软硬件结构复杂,导致处理速度和运行速率降低了^[5]。

为了优化控制算法,研究分析了多种智能控制算法:专家控制、神经网络、模型预测控制后,最终在 STM32 控制器上详细实现两种智能算法。智能控制算法可应用于各行业各岗位,可优化类似自动智能化控制问题,既可以解决一般的过程变量控制问题,例如温控、湿度等简单的过程控制,也可解决自动驾驶路径规划、机器人行为控制等复杂任务场景。以水箱液位密度控制为应用实例,两种智能过程控制算法的大致原理:首先是模糊 PID 控制,通过加入模糊控制器,把变量模糊定量化,然后用规则库推理出参数,完成自适应调节参数。并且还使用另外一种不同思路的智能控制算法:状态反馈控制。该算法根据模型建立状态方程,不同于传统的输出反馈,该算法可很好利用系统内部特性,方便直观的根据期望闭环极点直接获得优化因子。

要想优化核心算法、完成更复杂的任务,硬件平台需利用 STM32 平台,功能除了基本的获取各个模块的各种开关状态信息并控制输出模块、当液位超过安全阈值可以发出警报、关闭进水阀、关闭电源等功能呢,通信方面,上位机下位机可以互发数据,甚至实现一对多、多对多通信,上位机配有远程监控软件等。

以上分析了小型 PLC、经典 PID 控制的弊端，主要以改进智能控制算法为核心、同时改进硬件平台和软件架构、完善业务功能设计一套精确控制液位智能控制器。STM32 单片机具有高性能、低成本、低功耗的特性，可作为智能控制器的核心芯片，再结合企业压滤机生产车间的实际情况，设计合理的控制算法，不断迭代创新，真正做到了“从实际中来到实践中去”^{[6], [7]}。该系统的应用提高了企业效益、降低职工的体力劳动，使人民能做到生活和工作相平衡，践行了习近平总书记提出的“科技利民”、“为人民谋幸福”的理念。在物联网、人工智能发达的当今时代，煤炭企业的生存和可持续发展之路必须走上智能化、自动化。对于运输、装载、进料等设备进行智能化改造，辅以智能化调度，实现设备远程控制，设备自动化作业，水箱自动化选煤。总之，将嵌入式技术应用到企业生产环境中，提升工程设备的自动化水平，从根本上解决安全事故问题^{[8], [9]}。

1.1.2 课题提出的意义

该智能控制器优化了控制算法，同时具有丰富的功能，可用到企业实际生产中，以煤炭企业中的重介选煤工艺为应用实例。企业为了生存发展，在重介工艺改造不断创新。重介选煤对企业而言是一种有效且简易的方法，而重介悬浮液密度和液位的测量和自动调整对重介选煤工艺的准确读和有效性起着决定性作用，毫无疑问是最重要的环节^[10]。基于该控制器提出的一整套解决按方案有利的解决了企业的实际问题，既保障了职工的利益，又提高了企业的现代化水平，树立了企业有担当的积极形象。

对个人发展来讲，实际现场调研后对于企业需求充分了解后，站在宏观的角度进行架构设计、利用敏捷开发的思想，根据出现的问题及时迭代产品提升了自己的工程能力。在对重介悬浮液的液位和密度的控制算法研究中精益求精，尽可能利用多种算法实现精确控制，提升了自己的逻辑思维能力。作为控制算法不仅能控制介质桶内混合液的密度或者液位等变量，只要能确定对象的模型，可以举一反三的控制它的变量。对于自动化过程控制这一专题提升了自己的算法功底，有很重要的意义。

放眼未来，现在根据所学设计一套智能控制器，集成多种控制算法可满足多种控制需求，本文应用于煤炭分选设备过程控制中，充分的提高了效率。如今新能源还处于发展阶段，提高煤炭等不可再生资源的利用率非常重要。为了迈进新时代，我们需要研究学习各种智能算法和功能，并再次基础上厚积薄发，不断取得技术突破，当人工智能、物联网的技术突飞猛进时，该智能控制器会更加完善成熟。

1.2 课题研究的国内外发展现状 (Research status at home and abroad)

1.2.1 模糊控制和状态反馈控制的发展

智能控制算法有多种, 本文就实现的模糊 PID 算法和状态反馈控制算法进行介绍。

模糊控制算法, 是智能控制算法的一个重要分支。自从1974年英国的马丹尼工程师将模糊集合理论用于蒸汽发动机的控制以后, 历经 30 多年后, 模糊控制技术得到了普遍而飞速的进展^[11]。在机械手控制技术、太空飞行器的控制技术、家电智能化方面变现出了很大的使用价值。并且, 目前也有专门的模糊芯片可供使用。中国国内外对模糊控制器研究始于1979年, 现已在模糊控制器的概念、特性、计算、鲁棒性、电路实现方式、可靠性、以及规则自调整领域都做出了相当丰富的贡献研究成果^[12-15]。伟大科学家钱学森曾指出, 模糊数学的基础理论及运用直接关乎着中国在二十一世纪的国力发展与命运。

模糊控制语言能够定量描述人们的思想状况和复杂事物, 且相对于传统控制技术, 无需了解受控对象的模型、更容易对不明确系统或非线性系统实现控制、对受控对象的参数有着更高的鲁棒性、对外界的干扰有较强的抑制能力。

经典线性系统理论研究中单输入单输出线性定常系统研究也是可行的, 但其最显著缺陷就是只能说明输入输出的外部特征, 而无法阐明控制器内在的构造特点, 也就无法高效得解决多输入多输出系统问题。在 20 世纪 50 时代新兴的航天技术的带动下, 在1960年前后进行了由古典控制理论到现代控制理论之间的过度, 其中一个主要标志就是卡尔曼系统地把状态时间观念引进到控制理论当中, 而现代的控制理论就是正式地在引入状态空间与状态时间观念的基础上发展出来的。

现代控制系统理论中的线性系统里, 用状态空间法说明了输入状态与输出诸多变数之间的关系, 不仅反映了控制系统的输入输出外部特征, 也同时说明了控制系统内在的结构特征, 是一个既适合于单输入输出系统又适合于多输入输出系统, 既可用于线性定常控制系统也可用于线性时变控制系统的有效系统分析与综合方案。

现代控制系统理论中, 经常对系统设置状态方程, 以研究控制系统的能控特性与能观测特性。但状况反映方法不同于以往的输入输出反馈, 状态变量法可更全面的反应控制系统的内在特征, 因而较传统的输入与输出反映方法可更有效提高控制系统的特性。虽然状况反映方法不能影响控制系统的可控性, 却可以改善控制系统的可观测特性。只要原系统是能控的, 则一定可以通过正确选取反馈增益矩阵 K 来自由修改原闭环系统极点。而对于传统的状况输出反馈, 若不引入

附加的补偿设备，则这一点并不是总能实现的。因此状态反馈控制系统是现代控制理论中非常关键的组成部分，它在实际控制工程领域中也具有非常重要的战略地位。

1.2.2 悬浮液中自动控制的研究应用现状

国内外科研专家也已经就悬浮液密度、液位的自动控制提出了相应的研究理论和获取了丰硕的研究成果。

20世纪90年代，美国首先开发出了以经典PID技术为核心的自动控制系统^{[16], [17]}。后来又推导出了关于悬浮液密度液位控制系统的近似的数学模型，并给出了自适应控制的方案，结果得到了很好的控制效果^[18]。后来的研究也表明，以模糊PID控制为核心开发系统所得到的控制效率，比单纯的PID控制或单纯的模糊控制效率都要更好且适应能力也更强。但是这些研究都是针对单变量，而重介选煤过程至少有密度和液位两个变量，耦合性较强，于是设置了两个独立的模糊控制器分别控制液位和密度取得了较好的效果。各种算法发展改进方向一是提高对重介选煤系统建模的精确性，一方面提高算法本身的适应性。之后国内提出了PFC_PID控制算法，该算法可以对密度和液位双变量进行控制，对数学模型要求低，所以内环用PID算法稳定了系统，外环用PFC控制，以克服滞后问题，取得了较好的密度和液位控制效果^{[19], [20]}。PFC控制算法属于模型预测控制算法，提出于20世纪80年代，与其他模型预测控制、广义预测控制一样，具有预测模型、滚动优化及反馈校正三个基本特征。但与其他模型预测控制不同的是需要预先选择基函数，并把基函数的线性组合作为输入量的结构，而后经过在线优化确定这些基函数的系数，从而确定系统的未来输入。

而在线优化于最优控制的范畴，其相关研究成为了现代控制理论中的重要一环，在飞机、航空和工业生产过程管理等诸多方面均被广泛应用。目前求解最优控制问题的方式，大致有解析法、数值计算法、爬山分析法和梯度性法。解析法大致可分成两类：当控制无约束时，采用经典微分法或经典变分法；当控制有约束时，采用极小值原理或者动态规划。如果系统是线性的，性能指标是二次型形式的，则采用状态调节器理论模型求解。而梯度型法是一种解析与数值计算相结合的方法，包括共轭梯度法、拟牛顿法等。

1.2.3 嵌入式技术国内外发展现状

智能控制器采用的是嵌入式技术。嵌入式系统诞生于微型机时代，硬件和软件都在飞速发展。本文中选择了ST公司ARM Cortex-M4内核的32位高性能微处理STM32F407ZGT6。该晶片属于ARM的Cortex-M4系列，由英国ARM公司为ARM架构的处理器提供了相应的ARM处理器核心^{[21], [22]}，再由其他半导体公

司在此处理器内核的基础上设计芯片^[23]。STM32F407ZGT6微处理器，拥有最先进的内核、充足的资源和片内部设计、具备优质低功率的优点，可适应系统功能需求较多，并提供了十多种标准接口，能够很便捷的完成所有外设的设计与研发。资源丰富，主晶片自带 1M 字节的 FLASH，可外扩 1M 字节的 SRAM 和 16M 字节的 FLASH，能适应大存储器需要和大数据。板载新型音频编解码晶片、六轴传感器、百兆网卡、光敏传感器等各类接口技术芯片，满足用户了不同使用需要，目前已成为在工业控制中具备性价比的主要核心芯片。

目前除了这款新芯片，中国还在构建自主产品生态系统。龙芯作为一支科研型的团队发展到现在走过了三个成长时期。

第一个阶段始于 2001 年至 2010 年，这是龙芯团队的重要一个技术的积淀期，在这一阶段，龙芯致力于研发 CPU 技术；从 2010 年起，龙芯转型成为现代企业，积极与市场融合，逐步形成了大、中、小三大系统的 CPU 产业，并重点面对信息安全领域、普通电脑、嵌入式应用领域等三大市场走向；从 2014 年起，龙芯公司步入了高速度发展期，上下游客户群体中依托龙芯 CPU 技术的软硬件开发人才已超过了上万人，而龙芯的收入也持续二年提升了百分之五十。目前龙芯的产品销售主要是按照三种主要产品系统进行定位，即龙芯 1 号、龙芯 2 号和龙芯 3 号。当中龙芯 1 号是面对特殊应用领域或需要而量身定做的“小 CPU”系统晶片，龙芯 2 号是面对工控和终端类应用领域的中“CPU”系统晶片，龙芯 3 号则是面向桌面/服务器设备类应用领域的“大 CPU”系统晶片^{[24], [25]}。

互联网时代的嵌入式产品，不但给中国市场展示了良好发展前景，带来了创新的生命力，但同时也对嵌入式系统核心技术，尤其是应用软件科技也是一个挑战。重点涉及：日益增长的各种功能密度、方便的无线通信、轻便的移动应用以及多媒体音视频处理。所以，嵌入式应用系统软件的发展需要强有力的基本组成工具，以及操作系统的良好支撑。

嵌入式实时操作系统最初诞生于上世纪七十年代，但如今随着网络、5G 技术和人工智能的广泛应用，嵌入式实时操作系统也以其自身的优点，在各个方便都得到了不错的发展。它不但能够高效降低成本，同时设计简洁，使用方便，非常适合各类使用场合景^[26]。

嵌入式实时操作系统最大的优点就是专业化较强，亦即每一种芯片都是一种单独的操作系统，各个系统之间也可能存在联系，但自主性却更强。嵌入式实时操作系统是必须按照使用者要求和硬件功能单独完成设计的，所以它具备了专业化很强的特性，而不同类型的控制系统和设施则不具备通用性，特别具备了独立性和专业化^[27]。

突出的是：实时性。而即时技术里又包括了硬实时和软实时。硬实时需要系统在规则的时限里就必需进行运算，所以硬实时里系统不可以超时工作，而在软实时里边处理超时的结果也不会如此严重。在实时操作系统中，人们可以将需要完成的功能分类为几个各项任务，各个各项任务都承担完成其中的部分，而各个各项任务也都是一段非常简单的程式，常常是一段死循环。RTOS 控制系统：如 FreeRTOS、UCOS、RTX、RT-Thread、DJYOS 等，而 RTOS 控制系统的核心内容就是实时内核。

FreeRTOS 是一种可裁剪、可剥离型的多任务核心，而且毫无目标数约束。FreeRTOS 提出了实时操作系统所需要的全部能力，包含资源管理、同步通信、多任务通信等。虽然 FreeRTOS 是用 C 和编译来写成的，其大部分都是用 C 语言写的，但只有很少的和处理机有关的小部分代码才是用编译写成的，因此 FreeRTOS 架构简单，可读性也非常强大，与 Linux 系统相比，是一个轻量级的操作系统。而操作系统的相同之处，无非就是进程、线程间相互通信、消息队列、内存管理、信号量和互斥等机制，从这个角度看 FreeRTOS 无疑是当下最值得研究学习的嵌入式操作系统^{[28], [29]}。

今天，通过嵌入式系统所产生的企业年产值已经突破万亿，由于嵌入式系统把信息技术和互联网紧密联系，并带来了很好的人机界面，智能系统也开始深入到了人们日常生活的方方面面。

1.2.4 软件开发模式的现状

在软件开发智能控制器时也应用到了敏捷开发模式，下面讲述下软件工程模式的发展。在 20 世纪 60 年代，软件系统规模相对较小，以作坊式发展居多。由于当时硬件网络的发展且软件产品规模与复杂性增加，在七十年代造成了软件产品危机。在八十年代推出了完善的产品制造方法，以“流程中心”分阶段来管理软件工程，该模式也被成为“瀑布模型”，每一阶段都要达到最好，这样前期工作做的越完美，后续的工作量也小，在一定程度上减少了软件产品危机^[30-34]。

但由于九十年代软件失败的经历促使过程中进一步增加了束缚与限制，软件开发流程也日趋“重型化”，研发效能明显下降、速度也减缓。

1998 年巴利玻姆公司真正提出了软件设计的“螺旋模式”，刚开始可以只完成基础功能，项目规模很小，当项目被定义得更好、更稳定、风险可控时，逐渐展开。螺旋模型在很大程度上是一个由风险驱动的方法系统，因此在各个阶段之前和经常进行的循环之前，都需要事先做好风险评价。

进入 21 世纪以来，软件工程模式逐渐走向了轻量级发展，较为知名的模式有敏捷开发模式。其更关注于程序员队伍和行业专家之间的密切合作、线下的技术交流（比书面形式的文档更好）、不断提交更新的软件版本、更为紧密且自我组

织式的队伍、可以很好地应对市场需求而变动的代码撰写方法和队伍组织方式，以及更为重视在软件工程中人的角色。而敏捷设计与迭代型设计二者都主张以较短的开发周期完成软件产品，但是敏捷开发的周期可能更短，并且更加强调队伍中的高度协作。

ISO9000 国际标准将在原来八大准则的基础上，增加敏捷准则。2000 年，美国向军方软件开发标准委员会推荐迭代的软件开发优选模型。在全球影响最大的美国波多里奇国际质量奖也把敏捷视为核心的十一大准则之一。Google、intel、IBM、华为、腾讯等公司都在使用敏捷开发模式进行高效的软件开发。

1.3 课题研究的主要内容和结构安排 (The main research content and structure of this paper)

1.3.1 课题研究的主要内容

本文主要是对基于 STM32 的智能控制器的研究与开发。首先分析了企业中小型对 PLC 控制中的经典 PID 控制情况。查阅文献分析了多种智能控制算法的原理，决定以重介选煤过程为应用实例，实现基于 STM32 的智能控制器，在控制器上实现了模糊 PID 控制和状态反馈控制算法。智能控制算法需要根据采集的信息做出决策，采集信息也是检测水箱状态的过程。通过液位传感器将液位信息转化成电流信号，通过模拟量输入模块读取该值。中间涉及到液位和电流的对应关系处理，需要实地到现场考察这个对应关系以便我们可以准确的根据电流值获得液位值。采用 PID 模糊控制算法和状态反馈控制算法控制水箱液位和密度。输出一个控制量，去控制进料阀门或者进水阀门的开度，该过程采用 PWM 控制。根据控制量设置占空比从而使得 PWM 输出一个对应的电流，同时也对应这阀门开度，对应关系也需要实地测量后得出。

PID 模糊自适应算法是根据当前偏差值和当前偏差值的变化速度，用比例积分微分处理过后获得合适的阀门开度，当然 PID 的参数初始化时可以根据实际情况事先设定好，然后用模糊自适应算法每一周期改变参数即可。

状态反馈算法是事先有一个期望的二阶惯性系统，和原系统相互对照，由现代状态反馈控制的理论获得优化因子，用优化因子对偏差值和偏差变化速度做处理得到控制量，进而得到阀门开度。

并完成其他配套智能功能，需要进行需求分析、硬件和软件的开发。该智能控制器可以和上位机串口通信、上位机配有远程监控软件，需人脸识别登录进行监控现场。各个功能模块使用 FreeRTOS 进行任务管理。串口通信的效果是可在 LCD 屏上可查看控制曲线、上位机 QT 界面实时显示数据。对控制过程中产生的

数据可存储在 EEPROM 或 MySQL 中，对工程代码用 git 上传到云端托管。总之，该控制器从算法上和功能上都实现了自动化、智能化。

1.3.2 技术路线

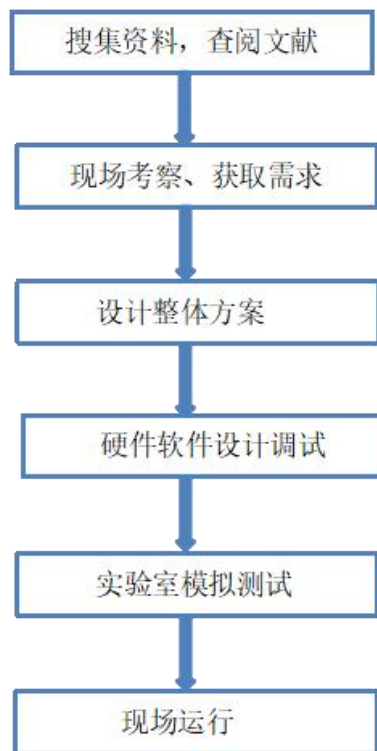


图 1-1 技术路线图

Figure 1-1 Technical route diagram

1.3.3 课题研究的结构安排

本文共分为 7 章，结构安排如下：

第一章为绪论。本课题提出了一款基于嵌入式 STM32F4 的智能控制器。首先阐述了课题的研究背景和意义，接着总结了国内外相关研发现状，制定了技术路线图，概括了论文研发的重点内容和整体架构安排。

第二是本课题所研究的需求分析与整体方案设计。该部分主要是对智能控制器的 PID 控制算法进行了改进，基于 STM32 控制器实现了两种智能控制算法。除了主要的算法外，硬件平台采用 STM32 控制器、软件架构采用 FreeRTOS 组织、除模拟量、开关量用控制算法外其他功能如通信等的设计方案。

第三章对各种现代智能控制算法的研究分析及选型。剖析了各种控制算法的基本原理、特点、异同点、实现方法等。

第四章则是该控制器的硬件设计。该部分针对企业的需要设计了硬件模块，详细对芯片工作原理和注意事项进行了说明，并焊接好模块后进行调试确保各模块均可正常工作。

第五章为该控制器的软件设计。主要是应用各硬件模块，包括利用 AD 采集电流、建立二阶惯性系统模型函数、PID 自适应算法控制和状态反馈控制算法，用 LCD 观察响应曲线、串口上传到上位机 QT 上。上位机使用远程监控，使用 MySQL 存储数据，并把整个工程代码文件等用放在了 gitee 云端上。

第六章为该智能系统的整体测试。主要包括智能控制算法的仿真测试、效果测试，工程问题如内存泄漏、死锁检测测试等。

第七章是本课题的总结和发展。这一节先对全部学术论文内容作了简短总结，分析了本课题研究过程中所提供的若干思路方案，并阐述了课题研究过程中的重点难点和论文中的缺点以及可改进的地方，同时还根据课题的科研方向，对今后的研究发展趋势作了展望预测。

2 智能控制器的需求分析

2 Demand analysis and of intelligent controller

本章根据现场调研进行需求分析，包括核心需求和其他需求分析，便于之后在分析基础上根据实践不断完善各功能模块。

2.1 控制器的核心需求分析 (Core requirement analysis of controller)

需求分析是一个项目设计与开发的基础。做出合理的需求分析有利于完成基于 STM32 的智能液位控制系统的开发，可以减少不必要的模块开发，确定我们的目标和方向。下面主要就控制算法的需求、各个通信协议需求、远程监控的需求、其他需求进行了详细的分析。

2.1.1 控制算法的需求分析

经典 PID 控制器还存在着一些局限，所以需要采用智能控制算法。获取水箱当前液位和目标液位的偏差 e 后，用经典 PID 控制时，比例积分微分的参数初始化后之后就不变了。但随着偏差变化，初始化参数还是保持不变就显得不合时宜。智能控制算法有专家控制、神经网络、模糊控制、模型预测、状态反馈等，在软件算法选型时，进行原理分析后，本文采用模糊 PID 控制和状态反馈控制在 STM32 控制器上实现。

表 2-1 传统和现代控制对比

	经典控制	现代智能控制
理论基础	经典控制理论	现代、智能控制理论
对象模型	确定、线性	不确定或非线性
任务要求	简单	较复杂

模糊控制，可根据每一轮的偏差和偏差变化速度进行修正 PID 的三个参数。这样可保证每个控制周期内 PID 控制器输出的控制量是最合适的。每次输入目标液位指令，系统都能智能调整参数，控制精度能得到保证，而经典 PID 只能保证首个周期的参数是适合的。

PID 算法本身是偏差控制，用 PID 控制器根据偏差得出一个最适合的阀门开度，优点是不受限于阶数、线性特性等系统本身特点，直接根据偏差进行控制。它的关注点是怎样对偏差进行最合适的处理，对系统是一视同仁，而状态反馈控制算法则反其道行之，直接把焦点放在对系统的处理上，要有一个期望的理想系统，从而获得优化因子，用优化因子和当前液位、液位变化速度、目标液位参与运算，继而获得本周期的控制量去控制二阶水箱。

2.1.2 通信协议分析

STM32 控制器获取到当前车间设备的参数信息后，可以很好的完成控制液位、密度任务。但有时多台设备需要联合控制去完成一个更大的任务，我们当然可以先让每台设备完成各自的任务，使它们有很好的自治性。然后人工将任务信息输入到另外的设备，依次递推完成整个车间的大任务。这个过程人工参与进来会带来诸多问题，影响工作效率、安全问题，违背了自动化、智能化、以人为本的初衷。要想使整个分布式系统提高运行效率、资源全局统一调度、不因某一结点故障而全体崩溃，必须选择合适的通信协议。嵌入式系统中常用的通信协议有 USART、IIC、SPI、CAN、TCP 通信等。本文中，上位机与 STM32 控制器之间使用串口通信传输数据，IIC 通信用于板间通信时距离较近，板间通信可采用 CAN 通信。远程视频监控系统使用的是 TCP 通信，通讯传输速度最高，并且通过级联设备能够扩展无限多个通信端口和扩展无限长的距离，相比较 CAN 通信而言传输速率和距离有关，是一种总线型的布线方案。

2.1.3 远程监控需求

除了基本的功能外，还需要远程视频监控现场作业，实现一个监控软件是必要的。采用人脸识别登录到该软件，然后可以打开现场的摄像头进行网络传输视频，监控设备的运行状况。

相比较而言，CAN 通讯的传输速率主要与距离有关，是一个总线式的布线方法，用这种布线方法查找故障点相对困难，而且因为通信电缆的原因，会影响到这条总线上各种电子设备的通讯传输，所以 CAN 通信是一个面向电子工程底层控制的通信网络系统。远程视频监控系统使用的是 TCP 通信，通讯传输速度最高，并且通过级联设备能够扩展无限多个通信端口和扩展无穷长的距离，是星型的布线方式，容易寻找到故障终端，其故障只影响到一个点，不会关联到其他房间设备。

网络通信的方式还包括 UDP 协议。适合大量数据传输，因为没有滑动窗口有和拥塞窗口的限制，传输速率更高。实时性高，没有丢失重传的机制，常用于竞技性游戏开发、直播。TCP 通信安全性更高、更可靠。TCP 通信是通过包的序号 seq、重排、超时重传的相结合的原理可以很好的保证数据的顺序，server 每收到一个包就启动一个定时器 200 ms，收到下一个 seq 包时刷新定时器，超时之后开始检测包的 seq，如果检测到顺序不对，之后接收的数据包全部重传。因为 TCP 通信协议的具有不错的优点，UDP 协议如果想多个 client 发送数据到 server 同时保证数据顺序正确，需要借鉴了 TCP 协议：给每个 client 分配一个 fd，这样 receive_buff 中的数据都分开了所以采用 TCP 通信。TCP 通信

是发展至今公认的最成功的计算机通信协议，为了确保数据传输的快速性、准确性，远程视频监控采用 TCP 通信是很合适的^[35-39]。

2.2 其他需求分析 (Other Requirements Analysis)

硬件分析了小型 PLC 控制的弊端，STM32 控制器作为 32 位硬件平台较为先进成熟，并且使用熟练度较高，所以以此为基础搭建硬件开发平台。

软件架构设计上为了协调各模块采用 FreeRTOS 系统，理论上比 ucos 管理的任务多，有时间片轮转调度功能，主要是开源免费。所以采用该操作系统进行统筹处理各个任务。

除了模拟量的输入输出，同时对其用智能控制算法调控外，开关量也值得重视。一些电闸意外断开，系统断电跳闸等信号需要被检测到，同时也可通过编程控制水箱系统的开关。实时获取的液位数据后，直观化的显示给工作人员，甚至画出响应曲线看系统是否符合快速性、准确性、稳定性的特点。还有其他业务功能：大量数据的持久化存储，既可以存储在 EEPROM 中，也可将数据制成表的结构上传到远程服务器上的 MySQL。工程项目代码、关键文件也可上传到 gitee 云端上，同时方便多人协作完成任务。

2.3 本章小结 (Chapter Summary)

本章首先详细分析了智能液位控制系统的需求，包括关键控制算法的优化、对开关量信号的处理、数据的直观化、持久化存储、工程代码的维护等，进行了需求分析以便了解开发的智能控制器的全貌，最终设计出一个智能控制器。该控制器支持模糊 PID 和状态反馈控制算法，硬件基于 STM32，软件设计采用 FreeRTOS 操作系统，功能除了基本的模拟量、开关量的输入输出控制外，还支持上位机串口通信、远程监控等配套功能。

3 控制算法的分析及选型

3 Analysis and selection of control algorithm

现代智能控制算法以不同的方法对经典 PID 算法的优化，各有特点、应用范围，以下是对常用的算法的原理、特点进行分析，包括：专家 PID、神经网络 PID、模型预测控制。随着计算机技术的飞速发展，以个人电脑为例，空间存储每隔几年就可以增大一倍且计算速度为每秒计算 $10^7 \sim 10^8$ 次，在完成复杂任务的基础上甚至还需考虑到时间复杂度和空间复杂度。

3.1 控制算法的分析 (Analysis of control algorithm)

(1) 专家 PID 控制

基本原理是不要求了解受控对象的准确模型，根据对于受控对象和管理规则的不同认识，并运用实际经验来设定 PID 参数^[40-44]。专家整定 PID 是将 PID 控制器与人工智能专家系统结合在一块，并按照工业生产流程的特点设定了控制器的技术指标要求，专家系统的技术指标确定方法是以误差信息为基准，再按照专家规则加以推演，从而决定了 PID 的技术参数调节方向与调整量，从而实现了调控功能。

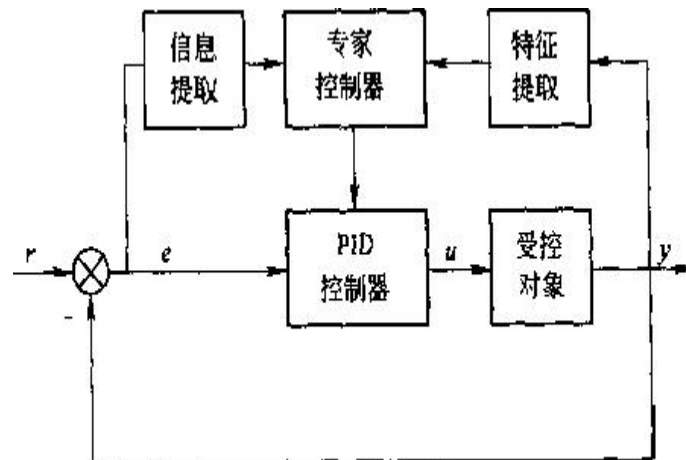


图 3-1 专家整定PID控制系统结构图

Figure 3-1 Diagram of Expert tuning PID control system structure

特征提取法：当系统发生扰动时，控制系统的实际输出将会有不同的波动性，例如衰减振动，等幅度振动，震荡发散等情况，专家控制器经过特性辨识环节，能够提炼出信息的输入输出或反馈曲线的特性信息，可能是超调量、峰值持续时间、衰减频率比、震荡频次、上升持续时间等，通过这些特性信息，专家控制器能够对 PID 参数做出调节。

规则获取：在此体系中，知识的获得主要有二个渠道，一个渠道就是能够从监控行业人员那儿得到测试经验，直接获得控制规则。目前，面对十几种的复杂

系统的反馈曲线控制研究专家已经调试汇总了一百多种规则，此外还有一个获得方式是通过系统仿真的方式能够间接的获得。

推理流程：通常都采用前向推理方法，推理流程可以分为动态过程或者静态过程，动态过程中涉及控制器的启动，扰动功能步骤和制动步骤等，专家控制器经过收集监控对象的输出量或偏差及其误差改变量、提取特征参量后，结合专家知识和经验，给出对应的 PID 调节参数的调整容量的幅度，从而实现了对控制器的参数调整流程。静态过程，是当计算机控制系统逼近于系统设计的平衡状态时，由专业人员以计算机控制系统中通过性能判断环节获取的预期的性能指标为基础，并从知识库中寻找 PID 整定参数的流程。

(2) 神经网络 PID

神经网络是一种以大规模并行分布式解决非线性问题的网络系统，能够以任意精度近似最复杂的输出有界的非线性函数，为人工智能控制系统开发提出了全新的路径^[45-49]。重要的神经网络模式有 BP 网络、径向基函数网络和于波网络等。

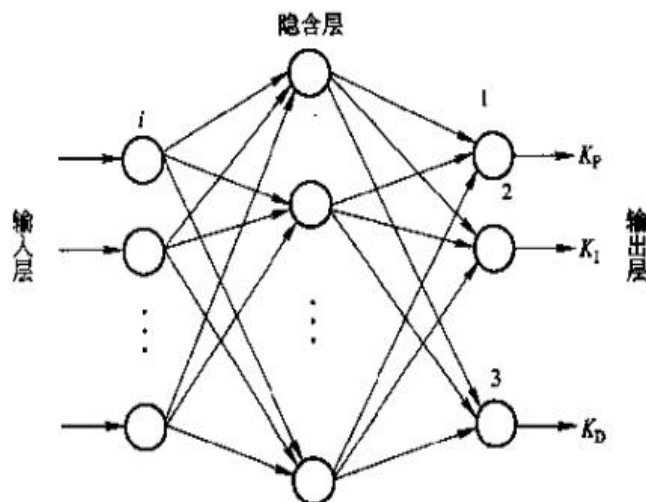


图 3-2 NN网络结构

Figure 3-2 Diagram of NN network structure

神经网络 PID 控制基本原理是通过定义神经元网络中 NN 的构成，即通过定义输入层节点数和隐含层节点数，得出各层次加权系数的初始值，并定义学习速率和惯性系数。对 e 和 ec 等变量统一管理，作为 NN 的输入输出，通过相关公式，估计 NN 各层神经元的输入与输出。最后确认神经网络的输入与输出 K_p 、 K_i 、 K_d ，进而根据增量的 PID 控制器公式，算出 PID 控制器的输入输出 u 。同时，神经网络经过学习会自动修正输出层和隐含层的加权系数，实现 PID 参数自适应调整。

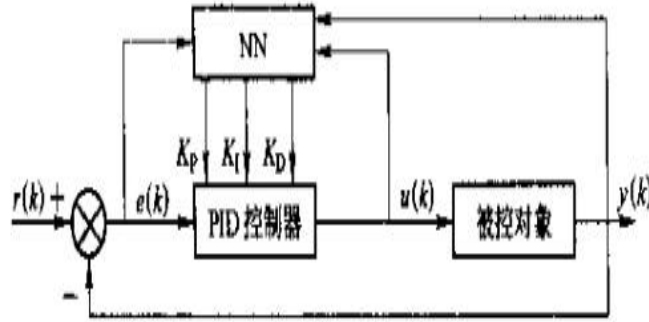


图 3-3 基于BP网络的PID控制系统

Figure 3-3 Diagram of PID control system based on BP network

(3) 预测控制

模型预测控制（MPC）根据系统模型推测输出未来输出，与PID控制相比，更加关注模型本身的特点，根据期望轨迹和未来的预测输出进行最优化控制输出，有一定的前瞻性。特点是主要使用状态空间分析法和最优性能指数设计方式，并要求准确的数学模型，反映了现代控制理论的发展趋势与特点、多应用于航空、航天等军事领域。与专家控制、神经网络控制原理不同，又为智能控制算法提供了新的思路。常用的控制技术有模型计算机控制系统（MAC）、以及动态矩阵控制系统DMC、多变量协调控制SFPC等^[50-54]。

模型预测控制系统在实现过程中的三个关键步骤，通常也被称之为三个基本原理，它们是预测建模、滚动优化和反馈校准。

预测建模：预期模型是模型预测监控的理论基石。其主要功用是根据目标的历史数据和未来输入，预计系统在未来的输出。关于预测模型的形式并不能作严密的限制，状态方程、转移方程这一类传统的模型，都可以作为预期模型。但针对线性稳态控制系统来说，阶跃响应、脉冲响应这一类非参数模型，也可直接作为预期模型应用。

滚动优化：模型预测管理可以根据某一特性指数的最优化来决定控制效果，但最佳优化流程并非每次离线进行，只是经过多次上线进行的。这正是滚动优化的基本特点，更是模型预测控制系统区别于传统最优控制的根本点。

反馈校正：为避免因模拟失配和环境扰动所导致的对理想系统的偏差，在新的采样时间就必须检查对象的实际输出，并使用该实时信号对基于模型的估计结果加以校正，之后再实施新的设计。

(4) 基于遗传算法的 PID 控制

遗传算法，是在一九六二年由美国 Holland 博士所发明的一个并行随机搜索优化方法，可以模仿自然科学界的生物遗传机制和进化规律。基于遗传算法的 PID 参数整定原理为：由 PID 三个参数组成基因型，由各技术性指标组成一定的适应度函数，通过使用遗传算法优选 PID 控制器，可以寻找并得到最优化的控制参量。尤其适合于解决非线性问题和复杂对象问题^[55-59]。

3.2 控制算法的选型 (Selection of control algorithm)

以上介绍了专家控制、神经网络控制、模型预测控制，各个智能控制算法都是根据系统的当前动态和静态特征，结合期望值，对 e 和 ec 进行处理运算，获取 PID 控制输出进行下一周期的调节。现代智能控制算法和经典控制的不同点如下表：

表 3-1 智能控制算法汇总
Table 3- 1 Intelligent control algorithm summary

	专家PID	神经网络PID	模糊PID
特点	主要依据现成专家经验	可表达任意非线性系统，需大量学习，速度慢、精度高	可模糊、清晰化数据同时结合模糊经验
控制方法	由专家规则直接改进PID参数	神经网络学习改进加权系数	模糊推理直接改进PID参数
应用场景	一般的控制过程、模型不明确，更注重稳态性能		模型较精确，注重动态性能

要想使智能液位控制系统获取高性能、高可用、高可靠的效果，必须设计合理的控制算法，经典 PID 算法应用非常常见例如 PID 算法控制电机、控制飞行器等等，代码较容易理解。在阅读了相关文献后，研究分析了多种智能控制算法的异同，在老师的指导下决定复现模糊 PID 自适应控制算法，由于本科自动化背景，对现代控制理论有一定理解，决定实现状态反馈控制算法。两种算法代表了两大类不同的理论和实践，却殊途同归，皆能优化控制。

3.3 本章小结 (Chapter Summary)

本章对智能控制器最重要的控制算法进行设计，对各个现代智能控制算法进行研究和分析，主要包括：专家控制 PID、神经网络 PID、模型预测控制、遗传 PID 控制，然后总结了异同点，最终决定以在 STM32 控制器上实现模糊 PID 和状态反馈控制算法。

4 智能控制器的硬件设计

4 Hardware design of intelligent controller

智能控制器不仅使用智能控制算法，还配套丰富的业务功能。本文以解决企业设备运行的实际问题，提出了基于 STM32 的智能控制器的研究与开发，其首要工作在于选择合适的硬件元件，其性能也直接影响着终端的运行效果和稳定性。为了降低开发难度、提高硬件运行的稳定性，挑选一款集模拟量、开关量输入输出、串口通信、CAN 通信、EEPROM 等功能于一体的嵌入式芯片是下一步开发的重要基础。本课题选中的 STM32F407ZGT6 作为处理核心，计算、存储能力相当不错，通讯接口多、可扩展丰富的功能，满足开发智能控制器的需求。

4.1 硬件部分总体设计 (Design of hardware)

为了实现这些功能要求，基于 STM32 的智能控制器的硬件总体设计如下图 4-1 所示，支持基本的模拟量和开关量输入输出和常规的 CAN、RS485 通信。

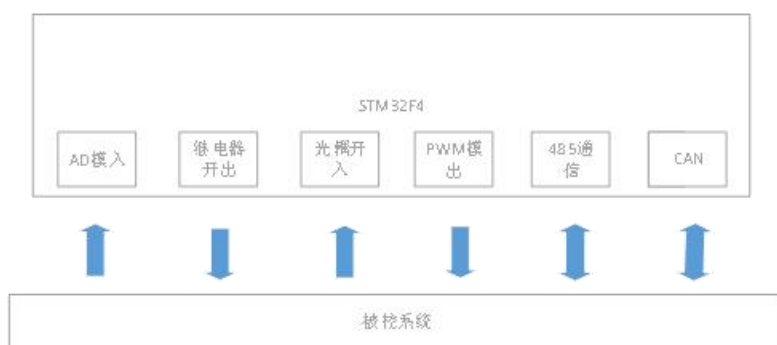


图 4-1 控制器硬件总体设计
Figure 4-1 Design of controller hardware

4.2 硬件部分各模块设计 (Design of each module of the hardware part)

4.2.1 电源电路模块

控制器需 3.3V 供电，本模块可以把外部 24VDC 转换成 3.3VDC，需要用到 TPS5403 进行转换得到。VIN 引脚连接外部 24V 电源电压，从 VSENSE 输出转换电压。在 COMP 引脚加上频率补偿组件 L、R 可以得到更精确的 3.3V 电压。ROSC 是开关频率控制引脚，SS 是软起动引脚，LX 是连接外部电感引脚。根据试验测试，最终设计了如图 4-2 示的电源电路原理图。原理图中，24V 电源右侧的滤波电容规格是 100 uf、25V 电压。可以选择值更大的电容，但是选择更小的可能会烧坏电容。

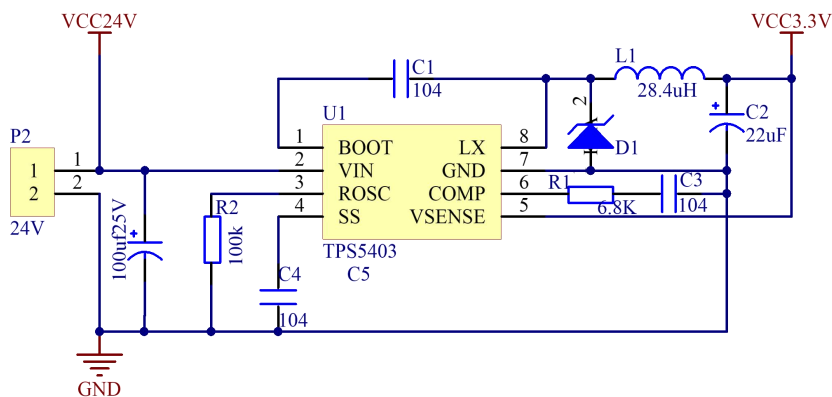


图 4-2 3.3V 电源原理图
Figure 4-2 3.3V power diagram

4.2.2 存储模块

STM4 开发板上的 EEPROM 芯片通过 IIC 总线与外部连接。IIC 通信是最简单的串行通信协议，可完成半双工通信。和串口通信不同的是，串口通信遵循 RS232 协议，有接受线 RX、发送线 TX 和地线 GND，由于收发是各自专用的互不打扰，是全双工模式，需要双方事先约定好比特率。传输距离比较近，高速 IIC 总线一般可达 400 kbps 以上。

在实际存储数据时，需要持久化存储液位高度、密度值、报警范围和其他相关系统状态，方便下次查看分析历史数据。24C02 芯片容量是 2Kb，可以满足需求。电路原理图如下图所示。把 A0~A2 均接地，对 24C02 来说也就是把地址位设置成 0。SCL 接 CPU 始终信号，SDL 是数据线。

IIC 协议使用时需要注意时序图，在传输数据的过程中总共有 3 种类型的信号：开始信号、结束信号和应答信号。其中开始信号是必须的，结束和应答信号都可以不要。开始信号：SCL 为高电平、SDA 由高电平转换成低电平，表示开始传输。反之，SDA 由低变高，表示传输完毕。应答信息为低电平时为应答有效。使用 AT24CXX_Write() 将字符串数组写进该模块，AT24CXX_Read() 从该模块中读取数据。

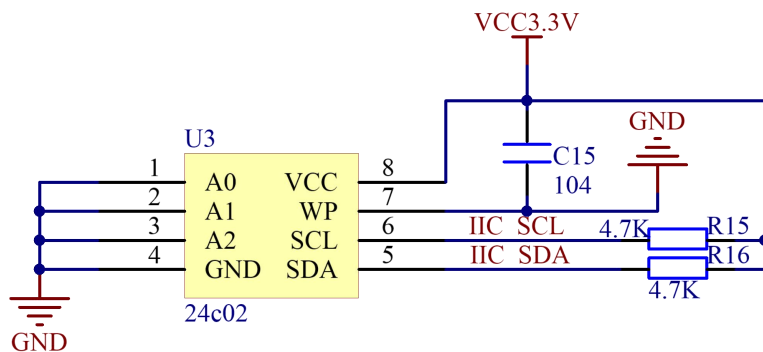


图 4-3 存储模块原理图
Figure 4-3 3.3V diagram of storage modules

4.2.3 LCD 显示模块

该模块支持 65K 彩色屏幕显示，分辨率为 320×240 并口，支持触摸屏操作，也是具有 FSMC 接口的，FSMC 亦即灵活的静态内存控制器，可以同时与同步或异步存储器和 16 位 PC 的存储器卡相连。

STM32F4 的 FSMC 接口提供支持包括 SRAM、NAND FLASH、NOR FLASH 和 PSRAM 等存储器。外部 SRAM 的端口通常有：地址线 A0~A18、数据线 D0~D15、写信号 WE、读信号 OE、片选信号 CS。

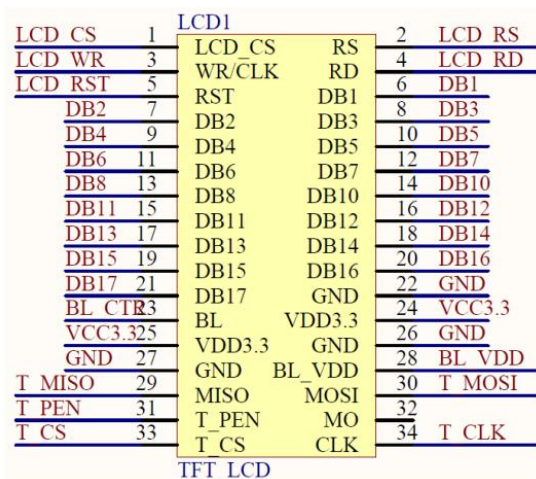


图 4-4 LCD 原理图

Figure 4-4 LCD diagram

4.2.4 模拟量输入模块

液位、密度传感器将高度、密度按照一定关系转换成电流信号，经过模拟量输入模块传到进入到 CPU 内部。外部电流经过滤波后输入到 STM32F4 内部的 ADC 模块，信号更加稳定。由于是使用内部 ADC，引脚 Vref 可以接上 3.3V 电压，不能接地，否则 ADC 模块将工作异常。ADC 模块是 12 逐次逼近型的模拟数字转换器。最大转换速率为 2.4Mhz，转换时间是 0.41 微妙（在 ADCCLK=36M，采样周期为 3 个 ADC 时钟下得到）。

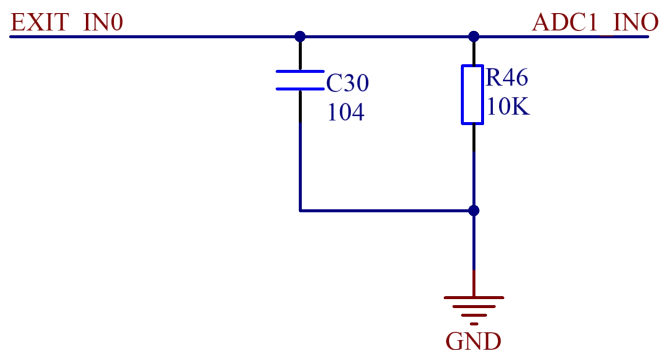


图 4-5 模拟量输入原理图

Figure 4-5 schematic diagram of Analog input

4.2.5 CAN 总线通讯模块

CAN 通信是一种能够实现分布式实时控制的串行通信网络。优点是传输速度快可达 1 Mbps，通信距离最远达到 10 km，用一个 CAN 端口即可实现简单通讯。通信具有不同的级别，可以使用 id 表示消息的优先顺序，使得最高优先权的 id 不会被中断。该模块对系统故障具有很强的鲁棒性，在复杂干扰性大的场合是一种理想选择，近年来价格也越来越便宜，应用较为广泛。

模块采用的 SN65HVD230 芯片完全兼容了 ISO11898 标准，内部集成了 CAN 收发器、控制器，将信号可靠的传送到 CAN_TX 端和 CAN_RX 端。

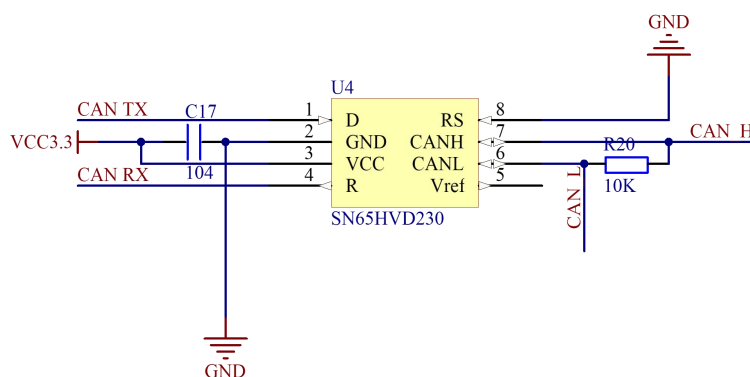


图 4-6 CAN 模块原理图

Figure 4-6 Diagram of the CAN bus module

4.2.6 模拟量输出模块

根据当前液位分析计算出控制高度从而得到数字量，再经过该模块转换成一定的电压电流值去控制阀门的开度。芯片采用 GP8102，MOSFET 模块的工作原理是在栅源间加正向电压，若之间电压大于开启电压时，漏源极导电，开始工作。软件编程需要用到 PWM 输出，通过修改占空比 TIM_SetCompare1() 函数输出一定有效值的电压。整个模块可将 0%~100% 占空比的 PWM 信号输入，线性转换成 4~20mA 的模拟电流输出。

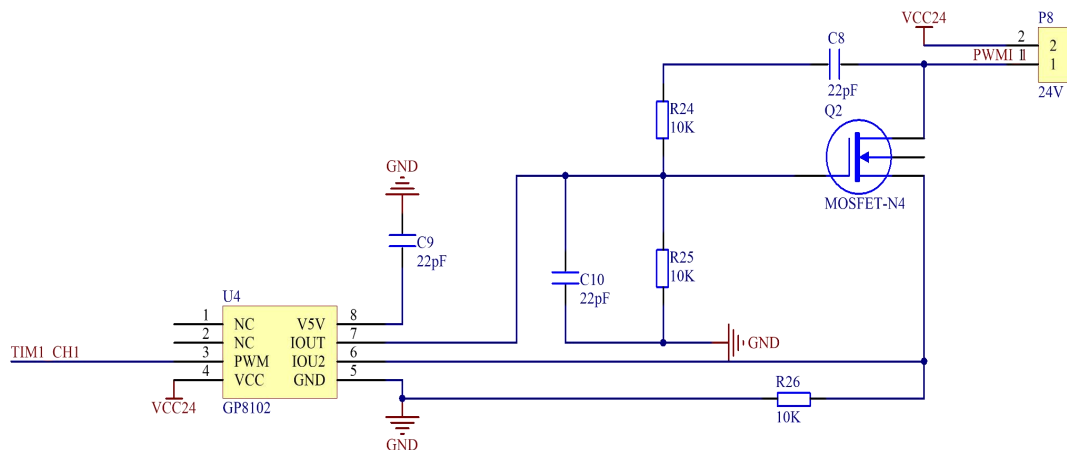


图 4-7 PWM 输出模块原理图

Figure 4-7 Diagram of PWM output module

4.3 本章小结 (Chapter Summary)

本文重点讲述了智能控制器的硬件部分的设计工作，主要涉及到存储模块、LCD显示器模块、模拟量输入输出输入模块、开关量输入输出模块等。分析了各个模块的主要功能、主芯片工作原理、为软件提供的使用函数。经过测试后，硬件设计没有问题，可以实现基本功能。交付厂家生产出实物图，效果如下。

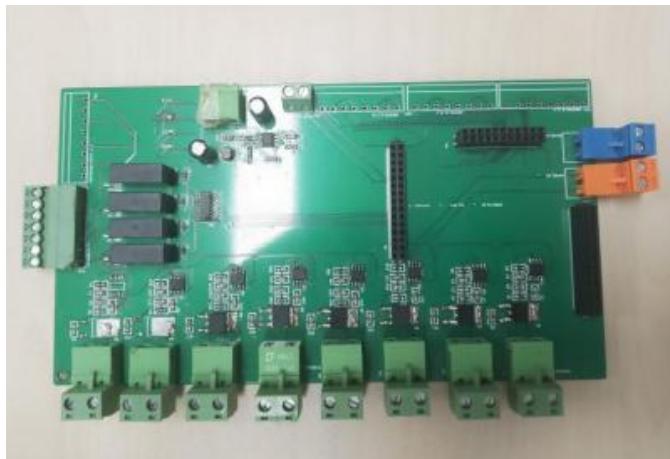


图 4-10 控制器实物图

Figure 4-10 Physical picture of intelligent controller

5 智能控制器的软件设计

5 Software design of intelligent controller

软件设计是智能控制器的核心组成部分，对良好的应用软件设计工作有着无法比拟的重要性，它胜于进行应用软件要求、应用软件代码、应用软件维修等任何一种过程，这也正是产品设计的最大优势。一套健壮的程序系统必须有完善的总体设计，而软件工程设计又是构成复杂软件系统中不容忽视的组成部分。

软件系统的稳定性、可维护性、工作效能等是直接关乎控制系统成败的关键因素，在智能控制器软件设计流程中应重视的如下三个要求：

(1) 模块化设计：一个大的软件工程项目需要多人合作开发完成。在智能控制器的设计中，大体分成控制算法、效果显示、远程监控等模块。每个模块完成后只需把接口提供给下一模块使用即可，模块间遵循着高内聚、低耦合的特点。可以有效的将整体项目分而治之，大大提高了软件的可重用性、移植性。在实际编程中也可封装自己的函数以供日后使用。

(2) 软件的性能：需要综合考虑各种条件，既需要考虑高性能的实现方案，把系统的时间和空间进行极致的压榨，又要考虑交付周期、方案成本、实现难易等因素。每一个需求都要详细设计，但不应过度设计，应该抓住主要矛盾、去除掉冗余的不能带来额外价值的功能，同时要在和客户的沟通过程中确定软件设计方案。

(3) 软件的健壮性：使用敏捷开发的思想，利用代码托管工具可在发生错误时有效回滚。再加上在软件编写时采取了防御性编写的方式，因此能够迅速找出问题出现的地方，从而大大提高了软件系统的可靠性、可用性。在后续的软件开发迭代流程中，提高了软件开发的质量和效率。

5.1 软件部分总体设计 (Design of software)

考虑本课题的功能需求，其中包括对水箱建立模型、使用 PID 模糊智能控制算法控制液位、用状态反馈算法智能控制液位、绘制相应曲线、串口通信收发数据等。

5.1.2 使用 FreeRTOS 操作系统

是开源免费的操作系统，对标准 C 类型的数据类型进行了重定义。函数名包含了函数返回值的类型、函数所在的文件名和函数的功能。传统的处理器同时只能执行一个任务，可通过快速切换任务，使得内部给每个任务分配一定的时间片，从宏观上看多任务时并行执行的，这就是 FreeRTOS 操作系统的多任务执行模式^[60-65]。智能控制器主要有三个任务，收集液位传感器的高度信息、根据当前液位

和目标液位得出调整值、PWM 输出控制水阀门开度，因此可把他们设置成轮转调度的方式执行。

```
/* 创建任务句柄 */
static TaskHandle_t AppTaskCreate_Handle = NULL;
/*AD任务句柄 */
static TaskHandle_t AD_Task_Handle = NULL;
/* 控制算法任务句柄 */
static TaskHandle_t control_Task_Handle=NULL;
/* PWM输出任务句柄 */
static TaskHandle_t pwm_Task_Handle=NULL;

static void AppTaskCreate(void);          /* 用于创建任务 */
static void AD_Task(void* pvParameters);  /* AD采集任务实现 */
static void control_Task(void* pvParameters); /* 控制算法任务实现 */
static void pwm_Task(void* pvParameters);  /* PWM输出任务实现 */
```

5.2 智能控制器各功能模块的设计与实现 (Design and implementation of each power module of intelligent controller)

嵌入式系统平台建设工作已经完成，核心的控制算法采用经典 PID 基础上增加模糊控制、现代控制理论状态反馈控制两种算法进行液位的控制。下面逐步进行了该控制器的软件功能的设计和实现，同时该部分实现了大量代码，因此需要摘取关键函数进行分析。

5.2.1 对系统建模的实现

从场景中抽象出二阶系统的模型，可以加上延迟环节等其他环节。取得离散化方程，发现当前的水位高度与前两次的水位高度有关。

T_1 是水槽的时间常数， T_2 是阀门的时间常数。 T_3 为两者的传递系数。

$$\frac{d\Delta h}{dt} = V_1 = \frac{c(t) - c(t-1)}{t - (t-1)} = c(t) - c(t-1) \quad (5-1)$$

再进一步求导：

$$\frac{d^2\Delta h_2}{dt^2} = \frac{v_1 - v_2}{\Delta t} = \frac{c(t-2) - c(t)}{2} \quad (5-2)$$

受双容水箱的启发，得到二阶微分方程：

$$T_1 T_2 \frac{d^2\Delta h_2}{dt^2} + (T_1 + T_2) \frac{d\Delta h}{dt} + \Delta h = Kr \quad (5-3)$$

代入之前推导的公式 (5-1)、(5-2)，令

$$tem = \frac{T_1 T_2}{2}, \quad (5-4)$$

$$tem2 = T_1 + T_2 + 1, \quad (5-5)$$

得到最终离散化的二阶系统函数方程：

$$c(t) = (Kr + tem * c(t-2) + tem2 * c(t-1)) / (tem + tem2) \quad (5-6)$$

根据以上分析可以写出二阶系统函数实现，该函数可以自行设置时间常数 T、传递系数 K 两个参数，和（5-6）式相对应。

/*二阶惯性环节 + 延迟环节 函数实现 */

```
float second_element(float giveValue, float T1, float T2, float k) { // 时间常数T
    static float result;
    static float ResultValueBack = 0;
    static float lastVal = 0;
    float tem = 0;
    float tem2 = 0;
    tem = T1 * T2 * 0.5;
    tem2 = T1 + T2 + 1;
    delay_ms(10);
    result = (k * giveValue + tem * lastVal + tem2 * ResultValueBack) / (tem + tem2);
    ResultValueBack = result;
    lastVal = ResultValueBack;
    return result;
}
```

5.2.2 信号的采集和输出

STM32F4 的 ADC 是 12 位逐次逼近型的模拟数字转换器，可以使用 MCU 内部或者外部的 ADC 功能，可以调用 `Adc_Init()` 将 ADC 驱动配置好。

通过液位传感器和密度传感器将非电量信息转换为电信号，而 STM32 控制器则利用 ADC 端口收集这些信息后，将模拟量信息转换为数值测量信息。于是经过两次转换，我们可以得到液位和数字量之间的对应关系。在现场只要完成接线就可以通过 STM32 官方提供的 `Get_Adc_Average()` 方法获取 ADC 采样后的原始值，相当于电流值。控制算法是输入参数是当前液位、目标液位，输出是控制量。所以需要将原始的电流值转换成液位方便为控制算法提供输入参数。转换函数如下：

```
float adc_voltage2height(float voltage) {
    return voltage * k; //k是转换系数
}
```

控制算法输出控制量 u 后，根据控制量改变阀门开度。如何找到这两者的对应关系，可以先计算出控制量 u 的变化范围和阀门开度，理论上进行简单的

线性映射。可根据实际情况进行对应的映射的调整,使得 u 可以控制 PWM 输出 4~20 mA 电流,进而阀门运转起来。PWM 脉宽调制是指使用微处理器技术的数值输入输出来对仿真集成电路加以调控的一门十分高效的方法。以锯齿波为例,当 CNT 超过 ARR 自动重载值时恢复归零,并开始下一个循环的输出。修改 CCRx 值就即可 PWM 输入输出的占空比,修改 ARR 值,可以修改 PWM 模拟量输出的频率范围,这也是 PWM 输出电流变化的基本原理。下面是根据 u 控制 PWM 模块可以设置输出电流。

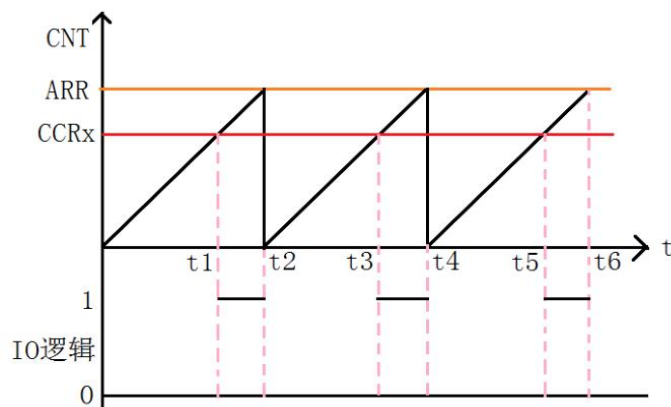


图 5-1 PWM工作原理图

Figure 5-1 Diagram of PWM working principle

设备的电闸的启停控制也是类似的,对开关量的输入和输出可直设置好 GPIO 口的工作模式为推挽输出模式。信号的采集检测和输出分析完之后,下一部分可以进行核心的控制算法实现,下面是 PWM 控制函数。

```
void pwm_realize(float u){
    TIM8_PWM_Init(500-1,84-1); //84M/84=1Mhz的计数频率,重装载值500,所以PWM
    频率为 1M/500=2Khz.
    TIM_SetCompare1(TIM8,u * k); //修改占空比
}
```

5.2.3 状态反馈控制实现

对于由受控对象与控制器两部分所构成的闭环体系,在经典系统理论中,采用输入输出传递的方法来实现系统。在现代系统理论中,可采取设置状态反馈方式来完成调节系统。经过设置状况反馈控制器,可以事先把系统朝着期望的稳定状态进行调整,到本周期的控制量,最终获取满意的控制效果^[66-71]。设计合理的 K 阵时,把整个系统极点设置到 s 平面上所期望的点位,过程如下,首先建立线性定常系统状态方程结构图:

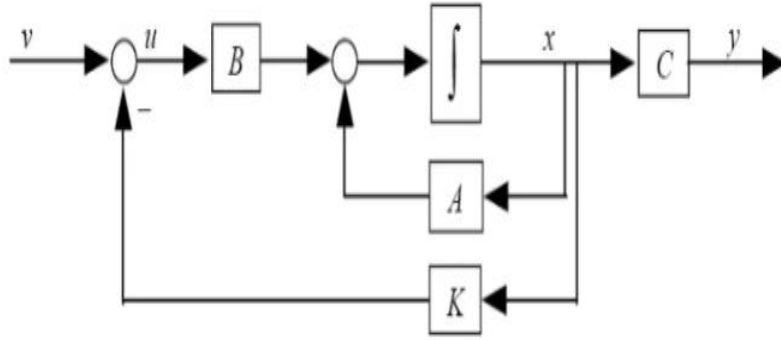


图 5-2 反馈控制系统结构图

Figure 5-2 Diagram of Feedback control system structure

状态方程 $\dot{X} = AX + BU$ 中， X 是状态， U 是输入（假设给定输入为1），状态反馈控制器： $U = -KX + R$ ， K 是状态反馈增益矩阵，根据之前对系统已经建立好的模型，可先带入设定的 T_1 、 T_2 、 K 、 r 等参数，可以得到参数 A 、 B 分别为：

$$A = \begin{bmatrix} 0 & 1 \\ -1 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}。$$

期望控制系统是稳定的，而线性系统稳定的充分条件是：闭环控制系统特性方程式的每个根都有负实部；或者说任何闭环传递函数极点都处于 s 左半平面。假设期望极点是 $S_1 = -1$ ， $S_2 = -2$ ，得到特征多项式：

$$S^2 + 3S + 2 = 0 \quad (5-7)$$

设待求的状态反馈阵 K 的诸多元为未知数： $K = [k_1 \ k_2]$ ，代入闭环系统的特征多项式： $f(s) = \det(sI - A + BK)$ ，可以得到 $K = [k_1 \ k_2]$ 的值。

软件关键代码实现：

```
curData[loop_n] = curHeight;
//求得控制量u
u = (matrixK[0][0] * curHeight + matrixK[0][1] * curSpeed) * (-1) + tarHeight;
lastHeight = curHeight;
curHeight = second_element(u, 1, 1, 1);
curSpeed = (curHeight - lastHeight) / 1;
```

可以看出，该控制算法没有用到 PID 控制，而是通过现代控制理论的方法获取优化因子直接处理。反馈增益矩阵 K 一旦确定后整个控制过程内不会改变，因此需要根据实际情况选取合适的期望系统得出较为合适的 K 矩阵。

5.2.4 PID 模糊控制算法实现

PID 控制器从问世起已经有将近 70 年历史。以其结构简洁、方便调控、稳定性好的优点非常广泛地应用于工程实践中。P 比例控制可以快速达到目标值，I 积分控制可以收集系统过去的偏差消除静差，D 微分控制可以调节系统变化的速度，有一定的预测功能。三者结合起来，发挥了很好的偏差控制效果。经典增量 PID 控制式：

$$u(k) = u(k-1) + K_p[e(k) - e(k-1)] + K_i e(k) + K_d[e(k) - 2e(k-1) + e(k-2)] \quad (5-8)$$

式中： K_p 、 K_i 、 K_d 分别为比例、积分、微分系数； $e(k)$ 为当前采样时刻的目标输出与实际输出之差； $u(k)$ 为当前采样时刻的控制量。

PID 的参数对系统的性能有着不同的作用：

比例控制 P 的意义：当比控制系统产生误差时，能够使控制数量朝着减少偏差的方向改变，比控制系统值越大控制功能就越强，缺陷是对具备均衡性的受控对象产生静差，虽然增大 K_p 值能够降低静差，但过大引起动态特性恶化，甚至使控制系统的振荡功能发散。

积分控制 I 的意义：对误差加以记忆和微分，可以减少整个控制系统静差。缺点是微分功能往往具用滞后功能，积分系数过大会使整个控制系统的动态特性变坏，从而损害整个控制系统的运动稳定性。

微分控制 D 作用：对偏差采用微分计算时，对偏差的变化比较灵敏，通过提高微分控制功能可提高控制系统的响应速度。缺点则是对干扰信号变化灵敏，反而使控制系统的抗干扰能力大大减弱。

综上所述，如果想要达到令人满意的控制器特性，使用最简单的 P、I、D 控制器都是不够的，尤其是对某些线性时变控制系统而言，在控制过程中需要根据控制系统的动态特征而选择合适的智能控制方法。例如变增益智能微分、智能微分和改变采样周期、时变参数等各种途径等，下面介绍模糊 PID 自适应控制算法。

模糊 PID 自适应控制算法的设计原理是在原来的 PID 控制算法的基础上，为了获得更精准的控制效果、为了能够灵活的调节 PID 三者的系数，特意增加了模糊自适应控制器，从而可使用模糊控制规则在线对 PID 参数设定调整，并以参数的调整量(ΔK_p 、 ΔK_i 、 ΔK_d)为输出，最终结合 PID 控制器中算出 PID 的输出，本周周期智能控制过程完毕，控制系统结构如下图：

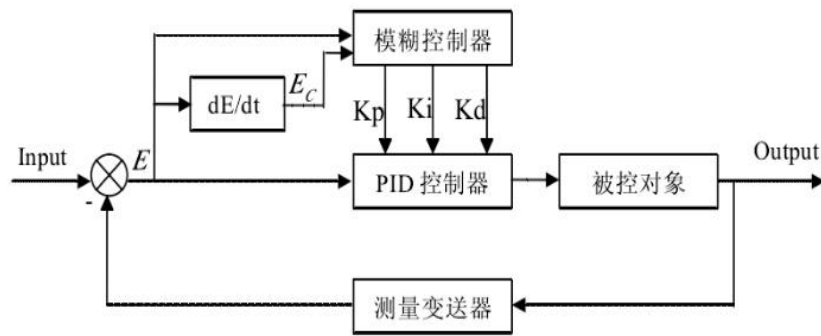


图 5-3 模糊自适应 PID 框图

Figure 5-3 Diagram of Fuzzy adaptive PID block

具体的设计方法：确定控制器的输入为二维输入 e ，把误差 e 和误差 ec 的变化率作为模糊控制器的输入，设计时也可以根据现场具体要求增加输入达到更多的维度。输出为 PID 参数的增量值 K_p 、 K_i 、 K_d ，输入和输出如下图：

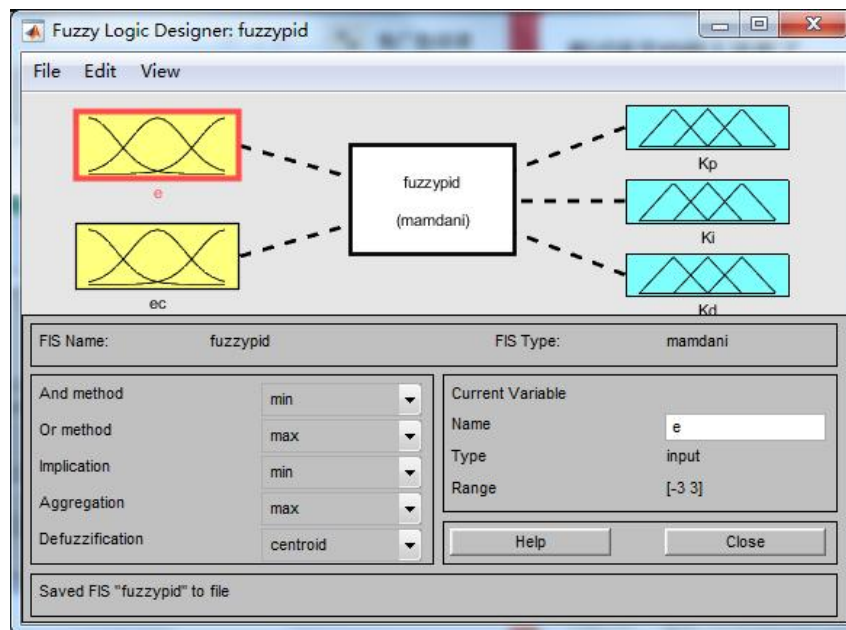


图 5-4 模糊 PID 输入输出图

Figure 5-4 Diagram of Fuzzy PID input and output

(1) 模糊化：对输入与输出的变量之间划定模糊区域，模糊规则主要是源于专家经验。根据不同的应用场景划分出不同的区间，首先确定论域的上下限，然后进行划分区间数量和长度，本文划分的模糊子集有 NB (Negative Big)、NM (Negative Megium)、NS(Negative Small) 等，并借此模糊规则表推理出 PID 的控制参数。

//dKp模糊规则表

```
static const float fuzzyRuleKp[7][7]={
    PL,PL,PM,PM,PS,PS,ZE,
    PL,PL,PM,PM,PS,ZE,ZE,
```

```

PM,PM,PM,PS,ZE,NS,NM,
PM,PS,PS,ZE,NS,NM,NM,
PS,PS,ZE,NS,NS,NM,NM,
ZE,ZE,NS,NM,NM,NM,NL,
ZE,NS,NS,NM,NM,NL,NL
};

```

如图所示是通过 MATLAB 仿真设置49条模糊规则：

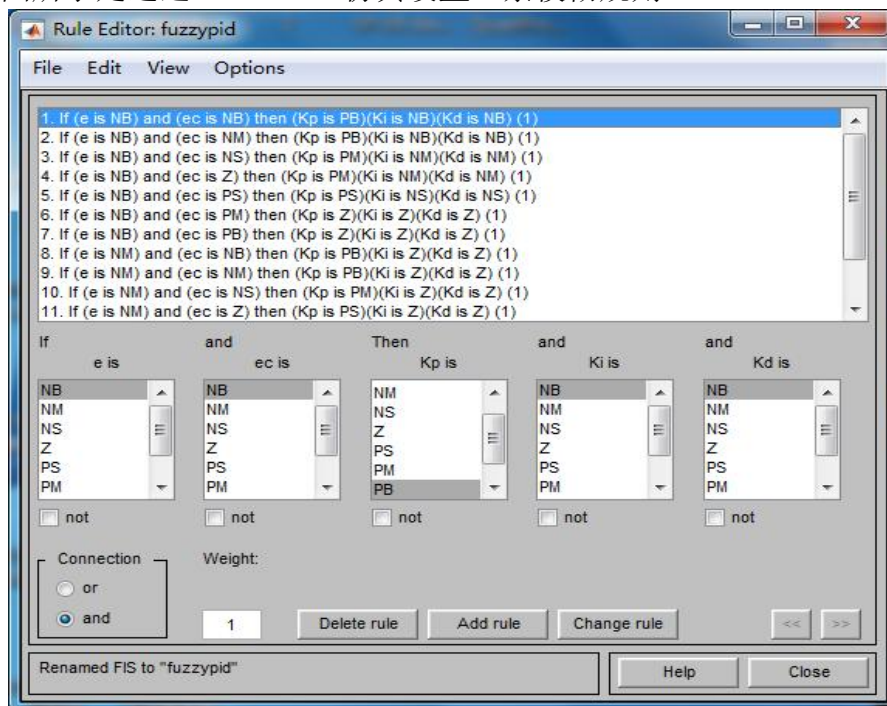


图 5-5 模糊规则图

Figure 5-5 Diagram of Fuzzy rule

(2) 模糊推理：根据模糊规则表决策出的模糊输出量。对采集回来的 e 和 ec ，求出其对应的隶属度，再通过模糊规则表求出比例系数 K_p 值所对应的隶属度。下面就是 C 语言实现的核心模糊推理函数 `Fuzzifier()`，该函数可以根据 e 和 ec 返回 PID 结构体的增量。

//根据偏差和偏差的变化

```

sPID Fuzzifier( sPID * tmp ){
    int eLeftIndex,eRightIndex,ecLeftIndex,ecRightIndex;
    float eLeftMs,eRightMs,ecLeftMs,ecRightMs;//隶属度
    sPID fuzzyDetPID;
    float e = 0;
    float ec = 0;
    e = tmp->e / (100*Ge);
    ec = tmp->ec / (100* Gec);

```

//先判断 e 和 ec 进行模糊化，放在左右两个整数区间内 如2.5，则 $eLeftIndex = 2$, $e_RightIndex = 3$


```

    eLeftIndex = (e/levelInterval)>3.0?3: (e/levelInterval)<=-3.0?- 4:
(e/levelInterval)>0?(int)(e/levelInterval): (int)(e/levelInterval)-1;
    eRightIndex = eLeftIndex + 1;
    //计算e对于eLeftIndex的隶属度
    eLeftMs = eLeftIndex<=-3?0: eLeftIndex==3?1.0: eRightIndex-e/levelInterval;
    eRightMs = eRightIndex>3?0: eRightIndex===-3?1.0: e/levelInterval-eLeftIndex;

    ecLeftIndex = (ec/levelInterval)>3.0?3: (ec/levelInterval)<=-3.0?- 4:
(ec/levelInterval)>0?(int)(ec/levelInterval): (int)(ec/levelInterval)-1;
    ecRightIndex = ecLeftIndex + 1;

    ecLeftMs = ecLeftIndex<=-3?0: ecLeftIndex==3?1.0: ecRightIndex-ec/levelInterval;
    ecRightMs = ecRightIndex>3?0: ecRightIndex===-3?1.0: ec/levelInterval-ecLeftIndex;
    //计算Kp需要四个Kp规则表中值
    fuzzyDetPID.Kp = (eLeftMs * ecLeftMs * DeFuzzy(eLeftIndex, ecLeftIndex, ID_dKp)
        + eLeftMs * ecRightMs * DeFuzzy(eLeftIndex, ecRightIndex, ID_dKp)
        + eRightMs * ecLeftMs * DeFuzzy(eRightIndex, ecLeftIndex, ID_dKp)
        + eRightMs * ecRightMs * DeFuzzy(eRightIndex, ecRightIndex, ID_dKp));

    fuzzyDetPID.Ki = (eLeftMs * ecLeftMs * DeFuzzy(eLeftIndex, ecLeftIndex, ID_dKi)
        + eLeftMs * ecRightMs * DeFuzzy(eLeftIndex, ecRightIndex, ID_dKi)
        + eRightMs * ecLeftMs * DeFuzzy(eRightIndex, ecLeftIndex, ID_dKi)
        + eRightMs * ecRightMs * DeFuzzy(eRightIndex, ecRightIndex, ID_dKi));

    fuzzyDetPID.Kd = (eLeftMs * ecLeftMs * DeFuzzy(eLeftIndex, ecLeftIndex, ID_dKd)
        + eLeftMs * ecRightMs * DeFuzzy(eLeftIndex, ecRightIndex, ID_dKd)
        + eRightMs * ecLeftMs * DeFuzzy(eRightIndex, ecLeftIndex, ID_dKd)
        + eRightMs * ecRightMs * DeFuzzy(eRightIndex, ecRightIndex, ID_dKd));
    return fuzzyDetPID;
}

```

(3) 清晰化：对模糊输出量通过清晰化方法判决出一个清晰量，选取重心法。

$$z_0 = \frac{\sum_{i=0}^n \mu_c(z_i) * z_i}{\sum_{i=0}^n \mu_c(z_i)} \quad (5-9)$$

z_0 是将模糊控制器输出量清晰化后的精度数值， z_i 是模糊控制量论域内的数值， $\mu_c(z_i)$ 为 z_0 的隶属度值。分子已经在模糊推理中求出，在本场景下分母为1，所以得到了本周期的增量 PID。pidController() 函数就是整个 PID 自适应控制算法的实现，根据本次控制的 e 和 ec 可以输出一个合适的液位控制量，通过 PWM 得到相应电压控制对应阀门开度。

```

dPID = pidController( tarHeight ,curHeight , &s);    //目标高度和当前高度
pid_out = (s.Kp+dPID.Kp)*s.e + (s.Kd+dPID.Kd)*s.ec + (s.Ki+dPID.Ki)*s.sumE

```

(4) 仿真实验。在 MATLAB 命令行输入 `sllookuptable` 命令，获得官方模糊 PID 的案例，以一阶离散系统：

$$\frac{K(z-1)}{Ts * z}$$

为例，输入为阶跃响应，搭建系统结构图。

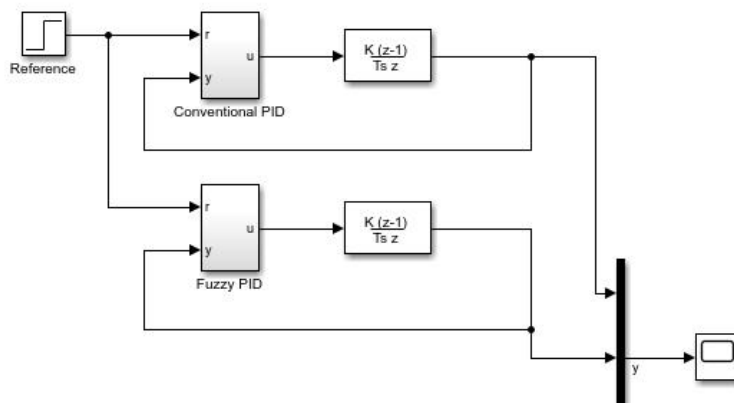


图 5-6 经典 PID 和模糊 PID 图
Figure 5-6 Diagram of Classic PID and Fuzzy PID

5.2.5 QT串口通信设计与实现

核心算法优化后，同时考虑到厂房里面具体的应用场景中，由于工作人员经常和上位机直接交互，为了方便观察数据，需要把关键数据发送到上位机软件上通知相关人员，同时工作人员也要通过上位机发送指令给下位机执行。可利用串口通信将液位数据从 STM32 控制器发送到上位机，上位机通过 QT 界面进行交互，其实该界面的功能相当于串口调试助手，可以设置串口、发送、接受数据。

QT 是一种跨平台的 C++ 库，主要用来开发图形界面程序，可在 Linux、Windows、Andorid、iOS 甚至嵌入式系统 VxWorks 上开发 QT 图形界面。QT 提供了很多模块，Qt Core，提供核心的非 GUI 功能，所有模块都需要这个模块，Qt Network 模块可实现 TCP、UDP 通信，Qt Multimedia 可提供音视频、摄像头服务，Qt Quick 可快速制作动画的界面，而 Qt Widget 应用非常广泛，是各个窗口的基类。QT 也提供了现成的类，QSerialPort 类是自带的串口通信类，支持访问串口^[72-77]，而串口通信的优点是只是用两根线通信、不需要时钟信号进行同步、有奇偶校验位进行数据校验、普及率很高，是最简单的一种通信方式，所以智能液位控制器采用串口通信的方式。

使用时在工程文件中添加 `QT+=serialport`，在头文件中添加 `<QSerialPort>`、`<QSerialPortInfo>` 库文件。具体的实现方法是：利用板上外设，可以采用定时器中断和按键输入控制的方式触发信号，将数据发送到上位机的 QT 上。用 QT 的 UI 模式事先实现一个类似串口调试助手的软件，确定好串口号之后，可以用 QT 自带 `QSerialPort` 类实现打开串口、接受、发送数据三个槽方法，分别由界面上相应按钮点击后触发信号获得相应即可^[78]。可用 `QSerialPortInfo` 打印出系统中所有的串口信息。构造一个串口对象并且用 `set()` 方法初始化配置，然后可用 `open()` 方法打开对应的串口。然后就可使用接收方法 `read()` 和发送数据方法 `write()` 完成串口通信的功能。

生成的界面如图所示：

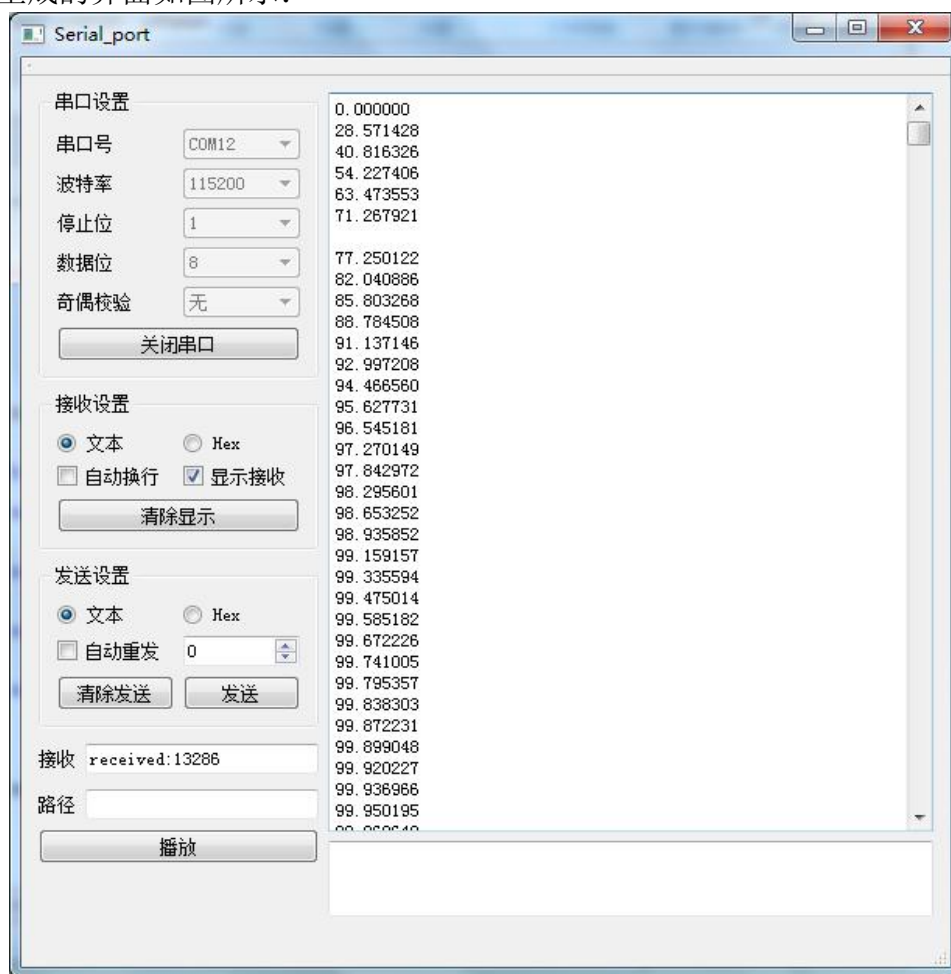


图 5-7 串口通信界面

Figure 5-7 Diagram of serial port communication

下位机通过按键将数据发送到串口，以下是 `send_data()` 按键选择发送数据的逻辑函数，发送数据通过循环 `printf()` 打印到上位机 QT 界面上。

```
void send_data(){
    KEY_Init();    //初始化与按键连接的硬件接口
    while(1){
```

```

key=KEY_Scan(0);    //得到键值
if(key){
    switch(key){
        case WKUP_PRES:    //控制蜂鸣器
            break;
        case KEY0_PRES:    //控制LED0翻转，同时打印数据到串口
            LED0=!LED0;
            for( serial_i = 0; serial_i < 300; serial_i++){
                printf("%lf\r\n",curData[serial_i]);
            }
        }
    }
}

```

上位机 QT 界面打开串口函数，里面包含对波特率、数据为、奇偶校验位、停止位的设置，以可读写的方式 `open()` 打开串口：

```

void open_serial(){
    QSerialPortInfo info_com1; //创建串口对象
    QSerialPort serial_com1;    //设置串口名
    serial_com1.setPortName("COM1"); //设置波特率
    serial_com1.setBaudRate(QSerialPort: : Baud9600); //设置数据位数
    serial_com1.setDataBits(QSerialPort: : Data8);    //设置奇偶校验
    serial_com1.setParity(QSerialPort: : NoParity);    //设置停止位
    serial_com1.setStopBits(QSerialPort: : OneStop); //设置流控制
    serial_com1.setFlowControl(QSerialPort: : NoFlowControl);
    serial_com1.setPort(info_com1); //打开串口
    int ret1 = serial_com1.open(QIODevice: : ReadWrite);
    if(ret1) {
        ui->PortBox->addItem(info_com1.portName());
        qDebug() << QString(" open serial_com1 success");
    }
    else{
        qDebug() << QString("t open serial_com1 failed");
    }
}

```

读取串口数据函数，首先是做些防御性编程，判断串口是否设置成功、接受缓冲区是否为空，并利用 `qDebug()` 方法输出调试信息。这里显示接受数据需要注意，缓冲区是最新数据，所以最终在文本框中显示的是已有的数据信息 `str` 和缓冲区最新的临时数据信息 `myStrTemp`。具体函数如下：

```

void Serial_port: : readData()
{
    QByteArray buf;
    if (serialPort){
        qDebug()<< QSerialPort: : Parity();//奇偶校验位
        buf = serialPort->readAll();
        qDebug()<<"buf: "<< buf;
        qDebug() << strerror(errno);
        qDebug() << "last error: "<< GetLastError();
        if (!buf.isEmpty())
        {
            qDebug() << "before ,receBytes: "<<receBytes;
            receBytes += buf.size();
            qDebug() << "after ,receBytes: "<<receBytes;
            QString redata = QString("received: %1").
arg(QString: : number(receBytes));
            ui->sendlabel->setText(redata);
            qDebug()<<"in function : buf: "<< buf;
            //支持中文显示
            QString myStrTemp = QString: : fromLocal8Bit(buf);
            qDebug() <<"after fromLocal8Bit: "<< myStrTemp;
            if(ui->reDisplay->isChecked())
            {
                QString str = ui->textBrowser->toPlainText();
                qDebug() << "first str: "<< str; //从文本框中获取的
                str +=myStrTemp;           //myStrTemp是buf中的
                qDebug() << "after + : str: "<< str;
                ui->textBrowser->clear(); //清空文本框中内容
                ui->textBrowser->append(str);
                buf.clear();
            }
        }
    }
}

```

5.2.6 画出响应曲线

画图的原理：智能液位控制系统可以实时采集每一个周期的液位高度，这样有助于实时获取控制效果并和目标高度作比较得到偏差量进行控制。在栈区开辟一个大小为 300 的数组存放浮点型的液位高度。根据该高度数组画出效果图。调用 LCD 自带的画点函数 LCD_writePoint() 自定义一个任意两点之间画直线的函数。首先在 LCD 屏幕上画出原点和坐标轴，再在 300 个点中相邻两点之间画直线，由于相邻两点距离较近，所以整体上形成了一条平滑的曲线。以下是画曲线的函数。包括画出横纵坐标轴、刻度、响应曲线。支持缩放，可以根据实际液位的不同自动调整刻度的数值，使得整条响应曲线都完美的呈现在屏幕中央。为了清晰的展现本部分的实现过程，用两个函数来说明画线的逻辑过程。

```
void showCur(float * curData ,int counts , float tarHeight)

//1. 先画y轴
for( ; yCounts < 400; yCounts ++){
    LCD_Fast_DrawPoint(30,450 - yCounts,POINT_COLOR); //原点（30，450）
}
LCD_Clear(WHITE);
drawLine(50, 50, 450, 50);
//2. 再画x轴
for( ; xCounts < 400; xCounts ++){
    LCD_Fast_DrawPoint(30 + xCounts 450 ,POINT_COLOR);
}
drawLine(50, 50, 50, 750);
//3. 这里的x, y 和最后画点时xy要反过来
yMax = (int)(tarHeight * 1.2); //获取y轴最大值
xMax = counts/10; //x轴最大值
//3.显示刻度
for( ; x_scale <= 4 ;x_scale ++){
    LCD_ShowxNum( 40 + (400/yMax)*(tarHeight/4)*x_scale ,25, (int)(tarHeight/4)
        *x_scale , 3 ,16,0);
    drawLine(50 + (400/yMax)*(tarHeight/4)*x_scale , 48 , 50 +
        (400/yMax)*(tarHeight/4)*x_scale , 40);
}
//4. 画出响应曲线
for( ; x < counts/10 ; x ++ ){
    tmpY = (int)( curData[x]*(400/ yMax) ) + 50;
    tmpX = (int)( x*(750/ xMax)) + 50;
    drawLine( lastY , lastX , tmpY ,tmpX);
}
```

```

        lastX = tmpX;
        lastY = tmpY;
    }
}

```

还有值得说明的是，先实现屏幕上任意两点之间画直线时也是由讲究的。因为具体画的时候是遍历两点中间所有的点，需要判断斜率 k 不存在、斜率 $k = 0$ 、 $0 < k < 1$ 和 $k \geq 1$ 的情况，根据合适的 k 选择截距是随着 x 轴还是 y 轴生长，进而可画图平滑的曲线。以 $0 < k < 1$ 为例， x 方向是每次画一个点， y 方向于 $\Delta x / \Delta y$ 处画点，此时 $\Delta x / \Delta y < 1$ 。用该方式画第二个点可保证两个点中间没有空隙，依次类推下去，接下来的点中间都无空隙。同理，当 $\Delta x / \Delta y > 1$ 时，应选择以 y 轴为单位画曲线。但是平滑的曲线只由合适的画图算法不能保证，还得保证合适的控制算法，使得液位从 0 变化到目标值的变化曲线符合快速性、准确性、稳定性的特点，尽量超调小、尖峰少，这才是本质上保证智能液位控制器的途径。

```

void drawLine( int x1 ,int y1 , int x2 , int y2){
    int x_counts = 0;
    int y_counts = 0;
    int x_abs = abs(x1 - x2);
    int y_abs = abs(y1 - y2);
    float a ,b ;           //a是斜率， b是截距
    int sx = MIN(x1, x2);  // sx 与 ex 需要成对使用 y与x不能混用
    int sy = MIN(y1, y2);
    int ex = MAX(x1, x2);
    int ey = MAX(y1, y2);
    int i = 0;
    int j = 0;
    if(x1 == x2){
        //垂直线
        for( i = sy; i < ey; i++)
            LCD_Fast_DrawPoint(x1 ,i ,POINT_COLOR);
    }
    else if(y1 == y2) {
        //水平线
        for( i = sx; i < ex; i++)
            LCD_Fast_DrawPoint(i, y1, POINT_COLOR);
    }
    else {
        a = (y1 - y2) * 1.0 / (x1 - x2); //y方向的斜率为  $\Delta x / \Delta y$ 
        b = y1 - a * x1;
    }
}

```

```

if( abs(a) < 1){
    for( i = sx;i<ex;i++) {
        LCD_Fast_DrawPoint(i,(int)(a*i+b), POINT_COLOR);
    }
}
else if( abs(a) >= 1){
    for( j = sy;j<ey;j++) {
        LCD_Fast_DrawPoint( (int)((j - b)/a) ,j, POINT_COLOR);
    }
}
}
}

```

关于绘图也可使用 QT 自带的 QPainter 类、QPicture 类、QPaintDevice 类，同时包含对应的头文件，再使用 showCur() 画曲线的算法在 QT 界面上直接画图。可使工作人员直接坐在电脑旁边监控液位变化情况，极大提升了工作效率。

5.2.7 汉字显示

STM32 自带显示英文字符和字符串的功能，而要想实现显示汉字字符串的功能首先要显示单个汉字字符，同显示英文字符的原理一样，只需获取单个汉字的字模，可使用 PCtoLCD 软件制作汉字字模。由于 STM32 控制器已经提供英文和数字字符显示的功能，所以就可以按照英文字符的显示逻辑进行实现即可：按照一定方向和顺序进行遍历每个点显示在 LCD 上。

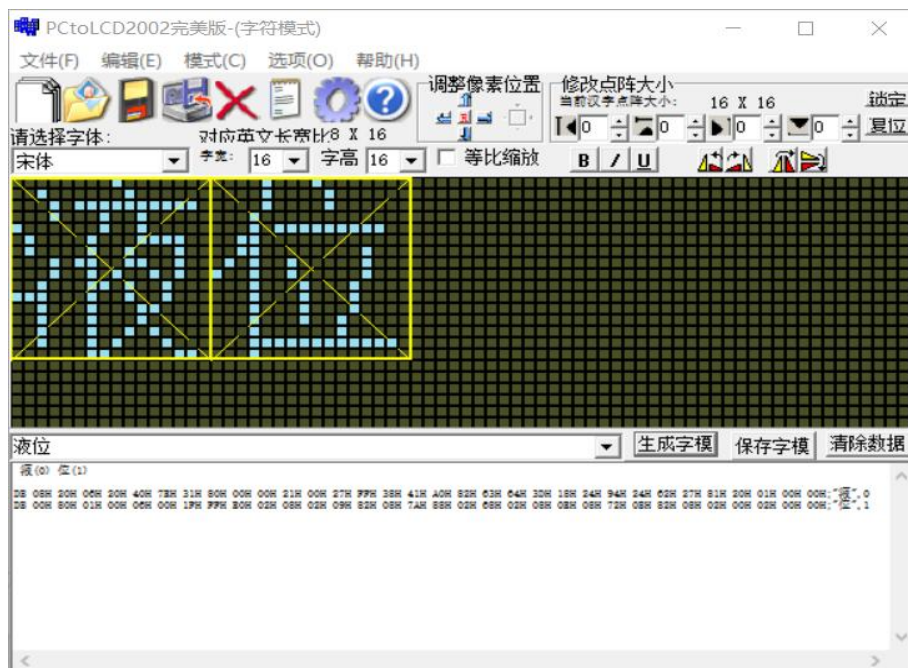


图 5-8 汉字取模图

Figure 5-8 Diagram of Chinese character generation font

具体汉字字符的实现逻辑，对每个汉字字符先找到字库里面对应的 32 个二进制数，根据二进制数在 LCD 屏幕位置上按照 x 和 y 的方向依次调用。LCD_Fast_DrawPoint() 方法画点即可，具体函数如下：

```
void show_chinese(u16 x,u16 y,const char *p,u8 mode){
    u8 k=0,wordByte,data;
    u16 y0=y;
    u8 wordNum; //字库字数
    for(wordNum=0;wordNum<3;wordNum++){          //字库里每个汉字逐一进行比较
        if(!strncmp(p,code_GB_16[wordNum].Index,3)){//目标汉字和字库里面的逐个比较
            //开始比较
            for(wordByte=0;wordByte<32;wordByte++){
                data =code_GB_16[wordNum].Msk[wordByte]; //得到目标汉字的32个
                二进制数
                for(k=0;k<8;k++){
                    if((data&0x80)==0x80)
                        LCD_Fast_DrawPoint(x,y,POINT_COLOR);
                    else if((mode == 0)) LCD_Fast_DrawPoint(x,y,BACK_COLOR);
                    data <<= 1;                //准备好画下一个点
                    y++;
                    if(y>=lcddev.height)return;    //超区域了
                    if((y-y0)== 16 ){
                        y=y0;
                        x++;
                        if(x>=lcddev.width)return;  //超区域了
                        break;
                    }
                }
            }
        }
    }
}
```

5.2.8 人脸识别注册登录

上位机软件设计成需要进行注册登录才能使用，有了注册登录的功能，才可以实现每个用户信息的私有化，进而区分不同用户的权限、对信息进行保密。项目负责人注册登录后可设置系统的初始参数、目标液位，软件开发人员注册登录后权限有限，可观察系统运行时的输出信息和历史数据，方便做进一步处理分析，同时注册登录后便可以使用远程监控功能。注册登录使用人脸识别的方式，而人脸识别采用 OpenCV 库实现。

人脸识别是一种生物特征辨识技术，已成功的运用在模式识别与图像处理等领域。2014年3月，香港中文大学讯息工程学院汤晓鸥团队推出成果，采用中国原创的人脸识别计算，精确度超过 98.52%，率先突破了人眼辨识水平。人脸检测定位是人脸识别的第一步，所进行的工作是将人脸从背景图片中检测出来，受图像背景、人的姿势、图片亮度等因素影响，人脸识别算法按照维度的大小分类可以分为二维和三位。二维人脸识别主要使用分布于人面部由低至高的八十个节点，通过检测人眼球、颧骨、上下颌等之间的距离，来实现身份认证。二维人脸识别技术方法的最大缺点，是在面临姿态、光线条件不同、表情变换和面部化妆变化等问题方面比较脆弱，因此辨识的精度受较大影响，但这都是由于人脸特征在自然状况下会随时显示出来的。因此三维人脸识别技术能够很大的改善辨识准确度，而真实的三维人脸识别技术通过使用深度图像的处理加以分析，从九十年代初期以来，技术已经有了相当的发展。

OpenCV 是一种开放的计算机技术视觉艺术和机器学习软件库，有多达二千五百种已完善的方法，既涵盖了古典的又涵盖了最前沿的计算机技术视觉艺术和机器学习算法，同时这种方法还被广泛用于测量和辨识人脸，以及追踪运动的物体，从而获得对事物的维建模，可以将图片拼接来生成一张高解析度的全景图像，从图像数据库中找到相似性的图像，本节人脸识别登录就使用到了 OpenCV 的人脸识别 SDK。先用 Qt 的 UI 模式设计界面，添加注册登录控件，双击添加信号和槽函数。可在定时器回调函数里用摄像头拍照获取第二照片和注册时的照片进行对比，比对成功可顺利登录，否则登录失败。点击登录按钮时触发的槽函数：

```
void COpenCVTmpDlg: : OnBnClickedButton2(){
    // 登录代码
    cap.open(0);
    // 创建定时器
    SetTimer(2, // 定时器的编号
            100, // 单位ms 1000/100 = 10帧
            NULL); // 表示时间到了之后去执行回调函数
}
```

首先调用 ASFDetectFaces() 进行人脸初步检测，对图片或者视频流中的人脸位置进行识别。ASFFaceFeatureExtract() 使用特定算法提取人脸的特征值，不是单纯的按照像素点进行对比。ASFFaceFeatureCompare() 进行两张照片比较，根据人脸特征值比对获得的相似度约在 0 和 1 左右，且相似度越靠近 1 越有可能是同一个人，该函数具体说明：

```

float faceCompare(MHandle handle, IplImage* img1, IplImage* img2) {
    MRESULT res;
    res=ASFDetectFaces(handle,cutImg1->width,cutImg1->height,
    ASVL_PAF_RGB24_B8G8R8, (MUInt8*)cutImg1->imageData, &detectedFaces1);
    if (MOK == res){
        SingleDetectedFaces1.faceRect.left = detectedFaces1.faceRect[0].left;
        SingleDetectedFaces1.faceRect.top = detectedFaces1.faceRect[0].top;
        SingleDetectedFaces1.faceRect.right = detectedFaces1.faceRect[0].right;
        SingleDetectedFaces1.faceRect.bottom = detectedFaces1.faceRect[0].bottom;
        SingleDetectedFaces1.faceOrient = detectedFaces1.faceOrient[0];
        res = ASFFaceFeatureExtract (handle,cutImg1->width,cutImg1->height,
        ASVL_PAF_RGB24_B8G8R8,(MUInt8*)cutImg1->imageData,&SingleDetectedFaces1,
        &feature1);
        // 单人脸特征比对
        MFloat confidenceLevel;
        res=ASFFaceFeatureCompare(handle,&copyfeature1,&feature2,
        &confidenceLevel);
    }
}

```

5.2.9 上位机实现远程监控

上位机采用人脸识别注册登录后，接下来使用 TCP 通信实现远程视频监控功能。只要保证被监控端现场和监控端在同一个网段内，便可实时监控设备运行情况。

被监控现场需要有另一台设备进行监控，可用 STM32 智能控制器也可用一台上位机代替，本文是用另一台上位机进行监控。需要提前配置好 OpenCV 图形库，然后就可使用 OpenCV 自带的摄像头功能进行现场拍照，之后通过 TCP 通信把照片发送给监控端即可。

需要包含头文件：<cv.h>、<highgui.h>、<opencv2/opencv.hpp>。

```

VideoCapture cap; //摄像头
void paiZhao(const char* fileName) {
    Mat frame; //OpenCV中表示图片数据
    cap >> frame;
    IplImage outImage = frame; //IplImage也是OpenCV中用来处理图片数据的
    cvSaveImage(fileName, &outImage, 0); //保存图片
}

```

TCP 通信容易引发分包和粘包的问题。

分包原因是发出端传送了总量相当多的数据信息，而收到端在读取数据时将数据信息分批送达，从而形成了每次传送的反复读写；分包的主要原因，是

由于 TCP 是以段 Segment 为单元发送数据的，形成 TCP 连接后，有一段最高消息长度（MSS）。一旦应用级数据包达到了 MSS，就会将应用级数据包分拆，并分成两个段来传输。这种时刻，接受端的使用层要拼接这二个 TCP 包，才能正确处理数据信息。

TCP 中粘包的现象通常是发送一端传输了一些数据，而接收端却一次读写了全部数据，从而造成多次转发却一次读取，其通常是通过网络流量优化技术，将几个较小的数据片段集满到一定的数据量，从而降低在网络链路中的传送次数。粘包的主要原因是 TCP 为增加网络的效率，会采用 Nagle 的算法，该算法表明，发送端即使有要传输的数据信息，如果待传输数据较少，将会延缓传输。这时应用层频繁的快熟传输数据，算法会将两个数据包粘到一块，最后只传输了一条 TCP 数据包给发送端。针对分包和粘包的解决方案是发送数据前，给数据附加两字节的长度。

整个流程是：先建立 TCP 连接，然后等待监控端发送监控指令，就执行 paoZhao() 函数拍取图片存放在工程的当前文件下。发送时最好有一个协议，先发送图片的长度然后每次发送图片文件的 4096 字节的数据直到将图片内容完全发送出去。附录 2 中的远程监控程序中监控函数 monitor() 里使用了 while 循环详细的展示了 TCP 完整接收端通信流程，具体步骤如下：监控端建立 TCP 获取被控端的 IP 地址、建立连接，发送监控指令后，就不停的从 socket 的 fd 中读取字节写到打开的 jk.jpg 文件中，用 count 记录读取的长度，当读取长度达到图片长度时为止。最后用 loadImage() 函数将图片显示出来，由于使用 while(1) 循环显示，就形成了视频，完成了远程视频监控的功能。

监控和被监控端在局域网内通信，通过路由器进行转接。在网络安全领域中，被控端上运行的程序可以被做成病毒强制运行，由于是采取 OpenCV 库操作摄像头拍照，没有调用操作系统的底层接口，所以很难被查杀，因此我们应该提高网络安全意识，不给病毒可趁之机。

5.2.10 存储在数据库中

智能液位控制器获得的数据既可以持久化存储在 EEPROM 中，又可以存储在上位机的 MySQL 数据库中。由于数据库具有索引和事务两大优点，所以本文介绍存储在数据库中的方法。可以租用各种云服务器，在上面用 docker 搭建 MySQL 环境，构建专属的云服务。MySQL 是目前主流的关系式数据库，跨平台性能强，不但可以在 Windows 控制系统上使用，还可以在 Linux、Mac OS 等控制系统上实现。MySQL 是完全开放的，所有人都可以修正 MySQL 的漏洞，也可以按照自身的需要来使用数据库。

因为数据库中的历史数据大多是根据表的结构进行保存的，从而可把智能控制器的热密度、液位、环境温度等信息，以表格的形式存入数据库中。因此需要根据数据的实际情况设计表的结构。数据库表的设计必须遵从以下三个范式。第一个是保证每一条的字段值都是不能解释的原子数。其二是在同一类数据库表中，仅存储了一组数据信息，并保证表的每一列都与主键有关。而第三范式则是保证表每一列都与主关键字的直接有关，而不是间接有关。本文以设备中的水箱设备号为主键，其他列为液位、密度等信息，通过 insert 操作逐条将数据插入其中。

MySQL 基本操作除了增删改查外，还提供存储过程、视图、游标等高级操作。一条 SQL 语句的执行过程时：先修改 change buffer，再写道到 binlog、redolog，返回给用户表示写成功了，最后开辟一个线程异步写到 FileSystem(B 树 B+ 树)中。从 Mysql8.0 开始有 query buffer，但后来的版本逐渐淘汰掉了，因为通过测试发现查找缓存命中率太低，而且查找缓存也很占用缓存的组件空间、根据性能考虑将其淘汰掉。同时由于 MySQL 的索引和事务两大特性具有不错的优越性，所以将其使用在智能控制器的持久化数据存储上。

(1) 数据库索引

智能控制器的数据可以随时通过 MySQL 进行查找。数据库之所以效率高，很大成都是采用了索引，相当于为数据增加了目录，提高了增删改查的速度。数据库的统计架构一般使用多路均衡二叉树 B- 树、B+ 树，但由于平衡二叉树在某种情形下会退化为链表，所以利用平衡二叉树 (AVL、红黑树) 就能够缓解这种问题，但由于平衡二叉树分支较少且磁盘 IO 较多，所以使用了多路平衡二叉树，其中序遍历是一个有序的结构。

索引的类别：主键索引、唯一索引、复合索引。

索引原则：对检索频次较高且信息量较大的表建立搜索，对字段选取使用频率较高，过滤效率好的列或者组合；采用最短搜索，节点所包含的信息最多，且较少在磁盘 IO 使用；最左前缀匹配原则；尽可能选取区分度最高的列作搜索对象；尽可能扩大搜索，在现有的搜索基础上，添加复合索引；尽量使用覆盖索引（只包含索引的查询），减少不必要的 select* 或者减少 select 不用的、非索引字段；

索引无效的特殊情形：对于 A or B，如果 A 和 B 中的一项都不包括在索引，则搜索结果无效；索引字段也不能参与计算；索引字段发生隐式类型转换；like 模糊查询以 % 开头导致将索引失效。同时索引优化的方法是：使用 show processlist 查看与 MySQL 所进行的连接是否锁表、查看连接状态、当前 sql 执行情况；explain 查看执行计划；trace 分析优化器执行计划；查看慢查询日志。

(2) 事务特性

事务的基本特点是原子性，访问更新数据库系统中所有数据信息项的一种运行单位，通过 `undolog` 进行回滚，其记录着事务的操作语句，回滚时进行具体操作的逆运算，`undolog` 还记录 `mvcc` 版本快照信息。

隔离性：每个读写事件的对象都与其他事件所操作的对象能彼此隔离，即在事件提交之前堆的其他事件均不可见。通过 `MVCC` 和锁实现。数据库中主要提供三个粒度的锁：表（`B+树`聚集索引）、页（`B+树`聚集索引的叶子节点）、行锁（叶子节点中某一段记录行）；`MVCC`主要解决一致性非锁定读的原理是碰到 `X` 锁就记录版本信息。

持久性，通过 `redolog` 记录具体数据。

一致性，一致性由原子性、隔离性以及持久性共同来维护的。

事务目的将数据库从一种一致性状态转换为另一种一致性状态。

5.2.11 代码管理

在开发智能控制器过程中，代码需要经过很多次更新迭代，可以将代码上传到云端 `gitee`（码云），防止电脑宕机等问题，也有利于多人协作开发。代码管理工具 `gitee`，是由开源中国提供的一个 `Git` 的代码托管服务。包括三个版本，社区版、企业版和高校版。相比 `GitHub` 而言，没有巨大完备的开发生态，因为 `GitHub` 发展较早，上面有种类丰富的工具集，甚至各种语言框架的核心开发组件也在 `GitHub`，但是其在国内的下载速度较慢、访问不稳定，所以采用 `gitee` 作为代码管理工具，将整个项目代码推送到云端的步骤如下：

- (1) 首先在 `gitee` 创建远程仓库，安装 `git` 工具
- (2) 在待上传的文件夹或所在目录右键单击 `Git Bash Here`，会弹出命令窗口
- (3) 把当前目录当作本地代码仓库，使用 `git init` 命令，会出现 `.git` 文件
- (4) `git add`，将文件添加到缓存区
- (5) `git commit -m “注释信息”`，添加注释信息
- (6) 本地和远程仓库绑定，上传到个人云端：`git remote add origin https://gitee.com/dkx-yongxin/dkx.git`
- (7) 拉取分支 `git pull origin master`，如果显示错误：`fatal: refusing to merge unrelated histories`，就是用命令强制合并：`git pull origin master --allow-unrelated-histories`
- (8) 把目标文件推送出去 `git push origin master`，弹出在 `gitee` 的账号密码输入即可，如下图是一次上传成功的图示：

```
Administrator@PC-20210326GALX MINGW64 /j/git (master)
$ git pull origin master
From https://gitee.com/dkx-yongxin/dkx
 * branch      master      -> FETCH_HEAD
Already up to date.

Administrator@PC-20210326GALX MINGW64 /j/git (master)
$ git push origin master
Username for 'https://gitee.com': 2833034152@qq.com
Enumerating objects: 31, done.
Counting objects: 100% (31/31), done.
Delta compression using up to 4 threads
Compressing objects: 100% (29/29), done.
Writing objects: 100% (30/30), 901.41 KiB | 1.66 MiB/s, done.
Total 30 (delta 8), reused 0 (delta 0)
remote: Powered by GITEE.COM [GNK-6.2]
To https://gitee.com/dkx-yongxin/dkx.git
d93773d..adbc29a master -> master

Administrator@PC-20210326GALX MINGW64 /j/git (master)
$
```

图 5-9 代码上传到云端图

Figure 5-9 Diagram of the code is uploaded to the cloud

5.3 本章小结 (Chapter Summary)

本章对软件设计进行了详细的介绍，包括对控制建立二阶惯性模型、PID 模糊自适应控制、状态反馈控制、发送数据到 QT、响应曲线的生成、汉字显示、人脸识别登录、远程监控、数据的持久化存储、工程项目的管理 git 等。

6 智能控制器测试试验

6 Test of intelligent controller

智能控制器的硬件部分经调试后最小系统可以工作，各模块无短路、断路现象、上电后电压、电流均符合要求，电路板长时间通电发热正常。下载测试程序都可实现基本的软件功能。软件设计和实现也是在硬件基础上进行应用开发，实现了丰富的功能。下面介绍通过 MATLAB 仿真得出的专家、神经、模糊 PID 等智能控制算法的优化情况，还有基于 STM32 的模糊 PID 控制和状态反馈控制算法的测试实验情况，另外在软件实现过程中，对工程中的常见错误内存泄漏、死锁检索进行测试实验和提出解决方案。

6.1 智能控制器软件测试 (Software test of intelligent controller)

6.1.1 控制算法仿真实验

(1) 专家 PID

用主要用 MATLAB 语言实现，控制对象为二阶系统，设置好专家规则后实时得出 PID 参数。如下图分别是无专家和专家 PID 控制下的响应曲线，专家 PID 控制误差响应曲线更加快速的在 0.05 s 内达到控制目标，整个动态控制过程平滑、波动更小，具有不错的优化控制效果。

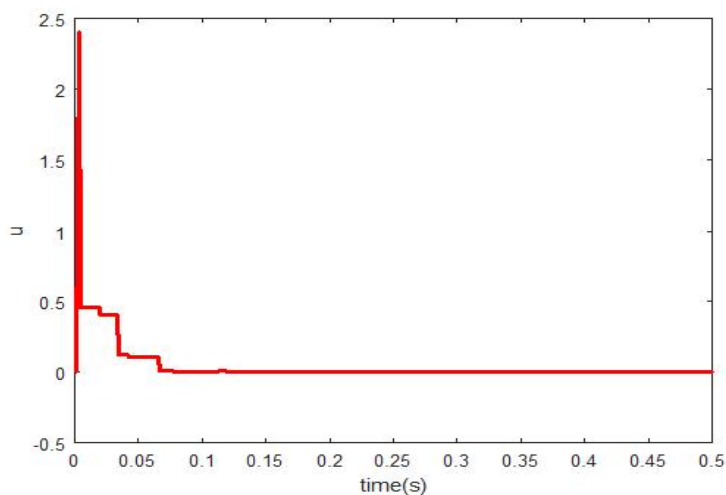


图 6-1 无专家误差响应曲线

Figure 6-1 Diagram of no expert error response curve

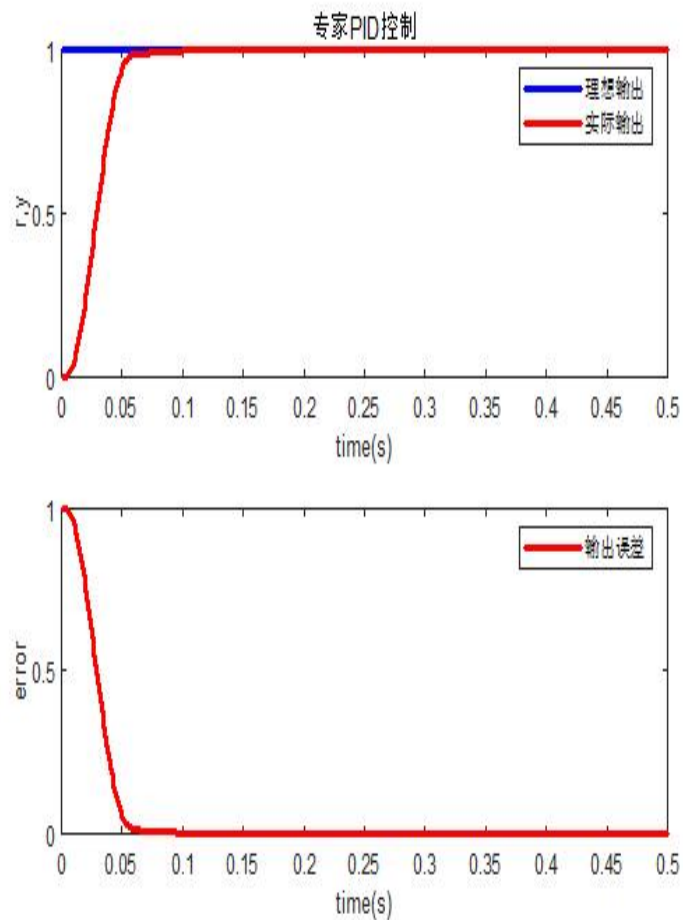


图 6-2 专家响应曲线和误差响应曲线

Figure 6-2 Expert response curve and error response curve

(2) 神经网络 PID

主要是基于 S 函数的 BP 神经网络 MATLAB 仿真。控制对象为二阶系统，首先初始化学习速率、惯性因子、隐含层、输出层的加权系数，最后，根据反向传播算法进行权值的调整，达到调整 PID 参数的目的。

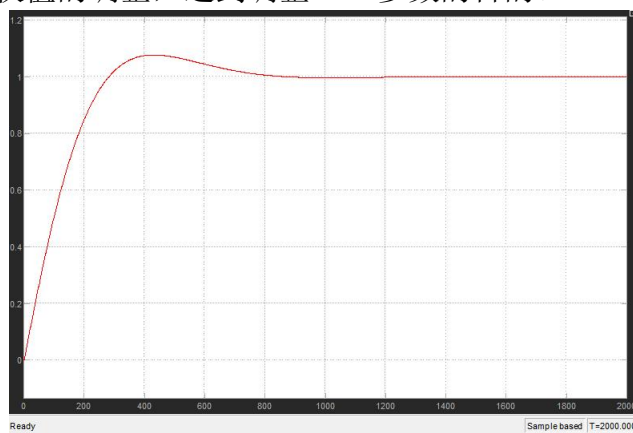


图 6-3 神经网络 PID 阶跃响应曲线

Figure 6-3 Diagram of Neural network PID step response curve

(3) 模糊PID

控制对象为一阶系统，蓝线是模糊 PID 阶跃响应曲线，快速性、稳定性都比经典 PID 要好。

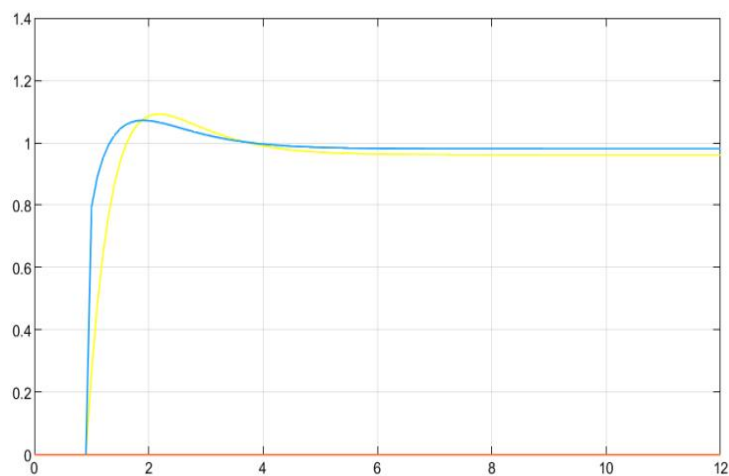


图 6-4 经典 PID 和模糊 PID 仿真结果对比图

Figure 6-4 Comparison diagram of classical PID and fuzzy PID simulation results

6.1.2 液位控制功能试验

在实验室的环境下测试，控制对象模拟的二阶系统水箱模型，控制变量是液位。显示放在 LCD 上，采用 PID 模糊控制后的效果曲线图，可以观察其动态性能。

开始的超调量不是太大，超调次数较少，调节时间、稳态误差符合要求。到达一定时间后，逐渐趋近目标液位，曲线非常的平滑，表明 PID 模糊自适应算法效果不错，画曲线算法也符合要求。

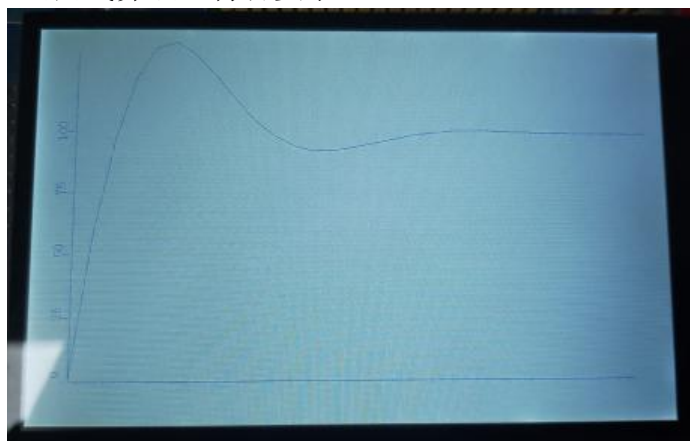


图 6-5 模糊 PID 控制 LCD 响应曲线

Figure 6-5 Diagram of Fuzzy PID control LCD response curve

采用状态反馈控制算法效果曲线图：可以看出响应曲线更加标准。类似一阶系统的变化量曲线，中间没有超调量，快速性不如 PID 模糊自适应算法，但最终的稳态误差较小，稳定性、准确性还不错，调节时间也比较短。



图 6-6 状态反馈控制 LCD 响应曲线

Figure 6-6 Diagram of State feedback control LCD response curve

关于两种算法的异同点见下表：

表 6-1 模糊 PID 和状态反馈汇总
Table 6- 1 Fuzzy PID and state feedback summary

	模糊PID控制	状态反馈控制
对象模型	对建模要求不高	需精确的模型
自适应性	在线调参	K矩阵不变，适应性不强
控制方法	根据e和ec调整模糊控制器	根据当前和期望模型调整反馈控制器
反馈	输出反馈可从外部直接测量，技术实现较简易	状态反馈，由目标任意配置极点、更有效改善系统性能

6.1.3 内存泄漏检测

内存泄漏的原因，在实现软件设计的过程中，由于动态分配堆上的内存而调用的 `malloc()` 函数，没有及时释放，导致系统在运行过程中，消耗尽 STM32 的内存。但是内存泄漏检测较为困难，当代码量足够多时，而 `malloc()` 函数的释放时机不同，最后给工程带来了不可估计的后果，因此需要找出合理的解决方案。

首先需要迅速定位到内存的位置，直接的方式有 windows 系统可直接查看任务管理器发现进程所占用的内存不断增大，Linux 系统通过 `top`、`htop` 等命令查看 CPU 占用率不断增大。可使用现成的内存泄漏检测工具，`mtrace`、`valgrind` 等，本文使用 `mtrace` 工具进行内存泄漏检测：

(1) 在 `main()` 函数的最开始包含一个函数调用：`mtrace()` 进行跟踪

(2) `export MALLOC_TRACE = test.log` 设置环境变量 `MALLOC_TRACE` 为一个文件名，存有内存分配释放的相关信息

(3) 运行程序，此时 `test.log` 文件会记录信息

(4) 输入 `mtrace a.out test.log` 命令，用 `mtarce` 进行分析

可搭配 `add2line` 工具使用，`addr2line` 开发工具是一种能够把指令的地址和可执映像转换为文档名、函数名称和源代码行数的开发工具。这在内核运行过程中出现崩溃时，就可以来迅速定位错误的位置，从而找到代码的 bug。

而内存泄漏工具的检测原理是：首先初始化钩子函数，在每次调用 `malloc()` 时，钩子函数被调用并录入下现场，最后通过检测堆内存分配链表以判断是否存在内存泄漏，并把内存泄漏的现场以可读的形式输出。但频繁的手动调用 `malloc()` 函数容易造成内存碎片不利于内存管理，为了从根本上解决内存泄漏，将泄漏的可能性降到最低，可使用合理的内存管理算法：

(5) 内存管理之伙伴算法

适用于物理内存以页为单位，不是以字节为单位。分配策略从 4k 重分配 8 bytes 后，将所有剩余存储器块进行按二的幂次进行分组，在将所有剩下的整块散成小块后，在所有空余的存储器中寻找比申请内存大的最小的内存块。当再次分配 4 bytes 时，查询剩余的 4 bytes 内存直接使用，如果没有将从 8 bytes 内存中分成两部分。一部分使用，而将另一部分的 4bytes 插入 4bytes 大小的链表中。回收内存释放的同时，系统还负责检查与之相邻并且同样大小的内存（简称伙伴）是否也空闲即：仅当两个内存大小相等且连一块时才可以被释放成一个更大的块。

优缺点：不用把内存分的非常细碎，回收条件苛刻，导致不能快速回收内存，并且每次至少分配一个页面 4K 浪费较大内存空间。

(6) slab算法

slab 分配器基于对象进行内存管理，`cache_chain` 的每个元素都是 `kmem_cache` 缓存。每个缓存存在 3 种 slab：`slabs_full`、`slabs_partial`、`slab_free`。

(7) Nginx内存池

Nginx 主要提供大块和小块的分配策略，两种结构体不同。小块一开始分 4K 内存，用 `last` 指针指向上次使用到的地址，用完之后一次性释放。分配大于 4K 的内存时用 `large` 指针进行管理，多个大块内存用 `next` 相连。

6.1.4 死锁检测

死锁，是指在各个线程和进度再执行过程中为抢占资源而形成的一个僵局，一旦进度和线程都陷入了这个僵持状况中，如无外力作用，它们将没法再往前推

进。线程 A 想获取线程 B 的锁，线程 B 想获取线程 C 的锁，线程 C 想获取线程 D 的锁，线程 D 想获取线程 A 的锁，从而构建了一个资源获取环。死锁的存在是因为有资源获取环的存在，所以只要能检测出资源获取环，就等同于检测出死锁的存在。在智能液位控制系统中，为每个任务分配一个线程，线程1的任务函数出如下，其他线程类似，就会形成死锁：

```
void* routine1(void* arg){
    pthread_mutex_lock(&mtx1);
    Sleep(1);
    pthread_mutex_lock(&mtx2);
    pthread_mutex_unlock(&mtx2);
    pthread_mutex_unlock(&mtx1);
}
```

死锁的危害：实际工程中开辟上百多线程是常见的，部分线程发生死锁占用系统资源不正常工作，直接根据不同版本代码检查逻辑很难发现是死锁的原因。因此实现一个死锁检测组件是必要的。详细的代码实现见附录1，组件实现的简要原理是判断死锁是否构成环，数据结构采用邻接表，节点用线程 id 和 lock_id 表组成，初始化钩子函数：

```
static int init_hook() {
    pthread_mutex_lock_f = dlsym(RTLD_NEXT, "pthread_mutex_lock");
    pthread_mutex_unlock_f = dlsym(RTLD_NEXT, "pthread_mutex_unlock");
}
```

在系统调用 pthread_mutex_lock() 和 pthread_mutex_unlock() 中加入三个原语操作 lock_before()、lock_after()、unlock_after()，在根据锁的两种状态实现这三个函数，在其中建立有向图，最后根据相关算法实现检测是否有环即可，下面是系统调用中的上锁函数。

```
int pthread_mutex_lock(pthread_mutex_t *mutex) {
    pthread_t selfid = pthread_self();
    lock_before(selfid, (uint64)mutex);
    pthread_mutex_lock_f(mutex);
    lock_after(selfid, (uint64)mutex);
}
```

检测环的算法是通过 DFS() 方法判断有环没环，每遍历一个点就将该点置为 1，表示已经检查过。最后不存在重复遍历的点就是没有环。

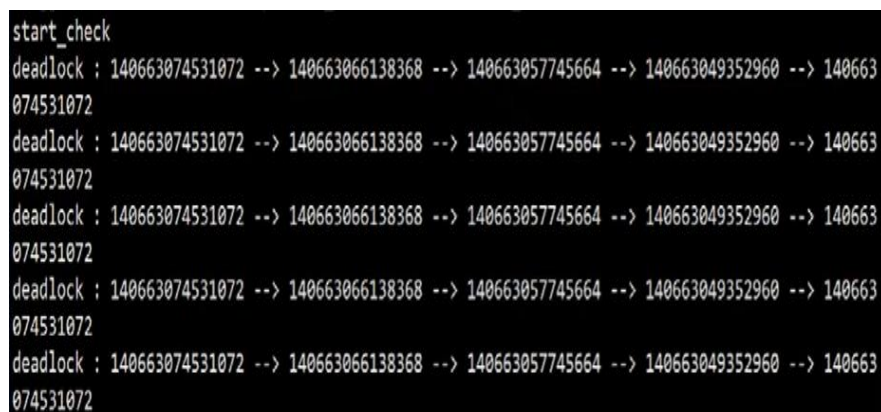
```
int DFS(int idx) {
    struct vertex *ver = &tg->list[idx];
    if (visited[idx] == 1) {
        path[k++] = idx;
```

```

    print_deadlock();
    deadlock = 1;
    return 0;
}
visited[idx] = 1;
path[k++] = idx;
while (ver->next != NULL) {
    DFS(search_vertex(ver->next->s));
    k--;
    ver = ver->next;
}
return 1;
}

```

最终效果图:



```

start_check
deadlock : 140663074531072 --> 140663066138368 --> 140663057745664 --> 140663049352960 --> 140663
074531072
deadlock : 140663074531072 --> 140663066138368 --> 140663057745664 --> 140663049352960 --> 140663
074531072
deadlock : 140663074531072 --> 140663066138368 --> 140663057745664 --> 140663049352960 --> 140663
074531072
deadlock : 140663074531072 --> 140663066138368 --> 140663057745664 --> 140663049352960 --> 140663
074531072
deadlock : 140663074531072 --> 140663066138368 --> 140663057745664 --> 140663049352960 --> 140663
074531072
deadlock : 140663074531072 --> 140663066138368 --> 140663057745664 --> 140663049352960 --> 140663
074531072
deadlock : 140663074531072 --> 140663066138368 --> 140663057745664 --> 140663049352960 --> 140663
074531072

```

图 6-7 死锁检测效果图

Figure 6-7 Diagram of deadlock detection

6.2 本章小结 (Summary of this chapter)

本章测试了无论是通过 MATLAB 仿真还是用 C 语言在 STM32 控制器上实现的软件液位控制的算法优化情况：根据响应曲线可以观察控制情况、得出控制算法的优化效果，控制器上的相关业务功能：数据存储、QT 串口通信、视频监控功能也已实现，还对业务功能中通用的工程中经常出现的内存泄漏、死锁检测进行测试，确保软件部分的准确无误。

7 总结与展望

7 Summary and Prospect

本文记录了智能控制器的研发过程，包括课题的研究背景、国内外现状、需求分析、硬件分析、软件设计、测试实验等，对设备的基于 STM32 智能控制有一定的借鉴意义。

7.1 回顾与总结 (Review and summary)

本文为了解决工业生产过程控制中复杂控制问题，具体以重介选煤液位过程控制为背景和应用案例，分析了企业的需求，针对这些需求，查阅相关文献、研究了多种智能控制算法，同时硬件平台在把原有小型 PLC 升级为以 STM32 控制器为载体。该控制器功能齐全，无论是针对核心的控制任务，还是相关业务逻辑开发，做到了有需求就有解决的方法。最终是结合实验室的条件完成了硬件和软件开发，完成了研发任务。主要的研究内容包括以下几个方面：

(1) 研究分析了经典 PID 算法的不足和多种现代智能控制算法的特点和应用，选用模糊 PID 控制和状态反馈控制作为优化改进算法；硬件平台分析了小型 PLC 的不足和 STM32 智能控制器的特点。软件框架利用 FreeRTOS 系统任务管理，功能除了基本的开关量和模拟量的输入输出控制外、还带有 QT 串口通信、远程监控、数据存储等等。

(2) 详细介绍了硬件模块的电路，研究了各个芯片需要完成的功能和工作原理，了解了硬件工作的基本配置、对协议进行了分析，为应用软件的开发提供的底层接口。

(3) 在硬件的基础上开发应用层程序，包括对系统建模模拟、AD 采集液位信号、两种算法控制和 PWM 控制阀门，效果用 LCD 和 QT 显示。人脸识别、TCP 远程监控、数据存储，各个模块之间采用 FreeRTOS 操作系统进行多任务管理。

(4) 经过 MATLAB 仿真和在 STM32 多次测试实验，表明智能控制算法具有不错的优化效果，并对软件工程常出现的内存泄漏和死锁问题进行检测修改、提出解决方案，以确保智能控制器的健壮和鲁棒性。整个控制器的开发过程，使我对智能控制有了深入全面的认识。

7.2 主要技术成果 (Main technical achievements)

7.2.1 设计了以控制算法为核心的智能控制器

不满足于一种控制算法，研究与分析了多种智能控制算法：专家 PID、神经网络 PID、模糊 PID、遗传 PID、模型预测控制、状态反馈控制。在 MATLAB 仿真工具上实现了部分算法，并在 STM32 控制器上实现了模糊 PID 控制算法和状态反馈控制算法。在重介选煤工艺的水箱控制应用实例中，模拟了水箱模型，只要设定好目标液位，算法就会给出最合适的进水阀门开度，使水箱自动达到目标液位且液位变化曲线符合快速性、准确定、稳定性的要求，说明智能控制器上的智能控制算法取得了较为满意的效果。

带有远程监控软件，使用当前上位机可以随时查看现场情况。直接使用上位机自带的摄像头，不用安装监控摄像头，方便快捷、省心不少。

养成了良好的工程习惯，对于工程最常见的内存泄漏和死锁检测问题进行了分析并提出了解决方案。

随着信息时代的来临，市场要求变动得更快，短时间交付也成为了企业的竞争力。轻量级的、更能应对市场变革的快速软件开发方式被市场广泛接受和普及。在设计智能控制器的过程中使用了敏捷开发的思维，遵从软件系统的客观规律，并持续的进行迭代与增量发展，最后提供满足顾客需求的软件产品，从而使得成本、生产力、服务质量、和满意度都有明显改进。

7.3 改进与展望 (The improvement and prospect)

本文针对企业的安全生产和信息自动化中的过程控制问题设计了智能控制器，智能控制器的目的是可以在控制器上实现多种智能控制算法，由于时间限制在 STM32 控制器上暂时实现了两种，后续还需完善。多种控制算法可以根据实际情况自动选用合适的算法，不需人为干预，若能自动识别阶数、线性非线性、建模难度等系统的特点，再结合对动态性能和稳态性能的目标要求，最后结合多种算法的特点，控制器可自动选用对应的智能控制算法，将对提高设备自动化和智能化水平有重大意义。状态反馈控制算法直接使用了手动计算出的 K 矩阵，实际上这一手动过程可交由程序完成，即设计成根据期望的二阶系统直接返回增益矩阵 K 。

对 FreeRTOS 系统的应用还不够深入，有很多优秀的机制：信号量和互斥、消息队列、内存管理没有落实到实践中去；很多业务功能的设计处于初级阶段，还需完善。

远程监控改进：被监控端暂时是用上位机进行监控，上位机的优势是更多是用来获取下位机的数据进行算法处理、图形化显示，可以弥补下位机的缺点。

所以可用 STM32 上的摄像头功能进行监控、采集图像；并且应用范围只能是在内网中，距离不能太远，如需实现超远距离监控，还要使用内网穿透技术或者购买专用的服务器进行转接以实现更远距离通信。此外除了远程视频监控，还可以增加录音功能、实现音视频的同步播放、将音视频信息上传到公网等等功能，总而言之，远程监控领域又是一片值得探索的新天地。

网络通信的改进：除了采用了图形化、持久化的方式，未来的智能控制器为了适应业务量增大、并发量提高的需求，可采用 `epoll` 机制、`libevent` 高性能网络框架；为了提高安全性，可以介入 `openssl` 库进行加密传输；为了提高传输速率，可对数据进一步处理，使用 `protobuf` 等应用协议。下面详细介绍下网络通信方面的改进和展望情况。

7.3.1 `epoll` 和 `libevent`

智能控制器和上位机之间或者多个控制器之间网络通信时，为了提高网络性能可采用 `epoll` 通信机制，同时运用 `libevent` 网络库。

`epoll` 的原理是：多条网络连接的情况下进行读写时用户态阻塞，此时由操作系统内核去监视读写的就绪状态，返回有事件的 `fd`。以 `read` 方法为例，对某个 IO 访问时，信息将会先被拷贝到系统内部的缓存中，之后才会由系统内部的缓存中拷贝到用户空间。所以当一次 `read` 动作产生时会经过两个阶段：进行数据准备、以及将数据信息从内核拷贝到用户进程中，使用 `epoll` 机制的好处是可同时监视多路 IO，提高效率。`epoll_creat()` 方法创建一个 `epoll` 对象，包括红黑树、就绪队列两个数据结构。`epoll_ctl()` 会把该条连接加入红黑树，同时这条链接和网卡驱动程序建立回调关系，当网卡驱动检测到该条连接上有事件发生时，把该节点拷贝到就绪队列中。可用 `epoll_wait()` 将所有就绪队列中的数据拷贝到用户空间后利用 `for()` 遍历所有就绪的事件。

`epoll` 和 `select`、`poll` 区别：都是同步 IO、都需要对就绪事件进行读写。`select` 和 `poll` 都会从所有事先设定的所有 `fd` 中无差别遍历，而 `epoll` 会遍历最近发生事件中的所有 `fd`、告诉我们出现的事件类型。优点就是不会因为 `fd` 数量的增多效率而降低，可以获得活跃的 `fd` 并得到事件类型。每个 `select`、`poll` 都会将 `fd` 集合从用户态向内部态拷贝一遍，而 `epoll_ctl()` 拷贝到内核中是由内核存储，然后每个 `epoll_wait()` 将不再拷贝到内核中。

利用 `mmap` 可以加快内核和用户态空间的消息传输。不管 `select`、`poll` 或是 `epoll` 都要求系统内核先将 `fd` 文件通知给用户空间，而怎样减少无谓的内存拷贝也很关键，在这点上，`epoll` 是利用内核和用户空间映射到同块内存进行的。

同时可以使用高性能网络库 `libevent` 提高网络性能，`libevent` 的优点：

(1) 利用 reactor 模式对最基本的读写或出错三个事件可以提前自定义回调函数，在事件发生后由 reactor 直接调用回调函数，可用 event_set()方法直接设置事件的回调处理函数。

(2) 对于输入输出可带的缓冲区，libevent 在常规事件回调的基础上提供了一个关于缓冲区的 IO 抽象概念，该抽象概念被称为 buffer event，因为其提供了自动填充和释放的输入输出缓冲区，用户将不再直接处理 IO，而是从输入缓冲区读取数据，再写到输出缓冲区。

(3) libevent 带有计时器的机制，可以主动控制在特定的时间间隔以后调用回调函数，使函数调用适用不同的场景。

7.3.2 openssl 加密

智能控制器产生的数据可用 openssl 库进行加密传输，openssl 功能多样而且开源，该库实现的功能有 SSL 协议、证书加密管理、大数运算等。C 语言为其主要开发语言，因此拥有良好的跨平台特性。

如 openssl 用于 https 通信的原理：openssl 生成证书（公钥、私钥、颁发机构、日期）给 server，浏览器发送数据后会拿到证书，把要发送的数据进行公钥加密，并和自己的对称加密私钥一起发出去。server 在获得数据后，会用自身的私钥进行解密，然后进行解析。解析完之后再利用 client 发来的对称加密私钥进行解密把加密后的数据给发过去。智能控制器可以给上位机生成证书、通信时可以采用 MD5、RSA 算法对数据加密传输提高了智能控制器的安全性能。

7.3.3 数据转换格式

控制器进行网络通信或数据存储时往往需要使用数据协议，了解数据格式的基本原理后，智能控制器在网络通信传输数据对原始数据进行转化，从而优化控制器的时间和空间效率。

正常的协议格式由消息总长度、加密的消息类型和数据组成。在通信协议最初的发展历程中，人们自定义了很多格式，常用的有 json、xml、protobuf 等。三种格式各有特点，主要是序列化速度和字节大小不同。protobuf 协议，是一个比 json 和 xml 更为轻量 and 效率的结构化数据保存格式，传输格式是二进制流但可读性不强，其必须通过 proto 文件查阅原文。常应用于 RPC 服务器、服务器之间通信等高速场景。xml 应用于本地，json 介于两者之间，即时通信用 protobuf 和 json 格式。json 是一个通用和轻量级的数据格式，具体来讲 json 常见的开源工具有三种：cjson、jsoncpp、rapid json。在使用上 json 有其独特的组织节点的方式：初始节点 rootObject 准备就绪可加进来对象、数组或者其他 json 值，同级别之间用

`next` 和 `pre` 指针指向，下一级子节点用 `child` 指针连接，所以 `json` 做序列化和反序列化的本质就是树的深度遍历和创建。

参考文献

- [1] 薛琼. 基于PLC的精选煤工艺控制系统的设计[D]. 内蒙古科技大学, 2019.
- [2] 马鑫. 重介质选煤过程自动控制系统的实践[J]. 机械管理开发, 2020, 35(02): 134-136.
- [3] 胡富月. 重介分选过程灰分自动控制系统研究[D]. 太原理工大学, 2021.
- [4] 田峰. 重介选煤加介工艺流程自动化改造[J]. 机械管理开发, 2021, 36(4): 200-201, 206.
- [5] 臧琼. PLC 控制优缺点探讨[J]. 内燃机与配件, 2018(14): 214-215.
- [6] 刘卫. 基于 STM32 的智能温度巡检仪的设计与应用[J]. 南方农机, 2022, 53(1): 149-151.
- [7] 安学武. 应用 STM32 系统设计基于物联网的农业小气候观测传感器[J]. 中国农学通报 2020, 36(16): 143148.
- [8] 魏东, 任毅. 酒钢西沟矿绿色智能矿山建设[J]. 矿山机械, 2021, 49(3): 1-4.
- [9] 蔡春雨, 陈志刚, 钟新荣, 等. 基于云平台的压裂车泵送系统监测设计与实现[J]. 国外电子测量技术, 2021, 40(10): 145-150.
- [10] 韦鲁滨, 孟丽诚, 程相锋, 等. 重介质悬浮液流变特性研究[J]. 煤炭学报, 2016, 41(4): 992-996.
- [11] 常昊天, 李小兵, 钟伟杰. 基于自适应模糊 PID 的二级倒立摆控制方法[J]. 火力与指挥控制, 2022, 47(2): 108-113.
- [12] 贾玉文, 段晓, 张厚明, 等. 研究堆 Mamdani 型模糊控制器设计优化方法[J]. 原子能科学技术, 2021, 55(6): 1091-1097.
- [13] AbouOmar Mahmoud S., Su Yixin, Zhang Huajun, Shi Binghua, Wan Lily. Observer-based interval type-2 fuzzy PID controller for PEMFC air feeding system using novel hybrid neural network algorithm-differential evolution optimizer[J]. Alexandria Engineering Journal, 2022, 61(9).
- [14] Wang Tingting, Wang Hongzhi, Hu Huangshui, Lu Xiaofan, Zhao Siyuan. An adaptive fuzzy PID controller for speed control of brushless direct current motor[J]. SN Applied Sciences, 2022, 4(3).
- [15] Wu Weiqiang, Gong Guofang, Chen Yuxi, Zhou Xinghai. Performance Analysis of Electro-Hydraulic Thrust System of TBM Based on Fuzzy PID Controller[J]. Energies, 2022, 15(3).
- [16] 莫程凯, 赵宇红, 段灵芝, 等. 两种群智能算法在 PID 参数优化中的应用分析[J]. 机械工程与自动化, 2022(1): 181-182, 186.
- [17] 王一夫, 王涛, 周鹏飞. 基于数据分析的打叶复烤流量控制方法[J]. 科技与创新, 2022(4): 54-56, 59.
- [18] 孙毅, 俞越, 单继宏, 等. 多股张力均衡的捻制装备自适应控制技术[J]. 计算机集成制造系统, 2021, 27(12): 3559-3569.
- [19] 郭西进, 邵宏清, 杨春宝, 等. 重介悬浮液密度与液位 PFC-PID 控制算法研究[J]. 工矿自动

- 化,2018,44(1): 89-95.
- [20] 于静,金秀章. 基于 RBF 神经网络的 PFC-PID 主汽温串级预测控制[J]. 华北电力大学学报(自然科学版),2020,47(6): 91-98.
- [21] 王春露,田瑞冬,赵旭等. ARM 处理器分支预测漏洞分析测评及新漏洞发现[J]. 西安交通大学学报,2021,55(7): 71-78.
- [22] 邹伟,喻俊志,徐德,等. 基于 ARM 处理器的单目视觉测距定位系统[J]. 控制工程,2010,17(4): 509-512.
- [23] 李沫,陈飞良,罗小嘉,等. 原子芯片的基本原理、关键技术及研究进展[J]. 物理学报,2021,70(2): 50-69.
- [24] 唐永学,朱桂梅,汤浩. 基于龙芯平台的无源检测装置设计与实现[J]. 计算机测量与控制,2021,29(4): 5-9.
- [25] 乔北. 我国高端芯片的发展机遇[J]. 电子产品世界,2017,24(2): 15-17.
- [26] 李亚辉. 计算机网络中的嵌入式实时操作系统[J]. 集成电路应用,2021,38(12): 142-143.
- [27] 王腾飞. 对计算机嵌入式实时操作系统的研究及分析[J]. 科技创新与应用,2020(36): 66-67.
- [28] 雷铭哲,张勇. Linux 线程机制研究[J]. 火力与指挥控制,2010,35(2): 112-114,118.
- [29] 周鹏,武延军,赵琛. 一种 Linux 安全漏洞修复补丁自动识别方法[J]. 计算机研究与发展,2022,59(1): 197-208.
- [30] 郭丁,孟令杰,于龙江,等. 高分多模卫星星地一体化专项工程研究与应用[J]. 航天器工程,2022,31(1): 24-30.
- [31] 冯东,齐国栋,唐宇. 新型信息化系统建设下的敏捷开发模式[J]. 计算机系统应用,2022,31(1): 91-98.
- [32] 蒋文婷,路子威,温颖,等. 航天研发项目流程管理与组合风险分析研究[J]. 航天工业管理,2022(2): 60-66.
- [33] 肖红军. 构建负责任的平台算法[J]. 西安交通大学学报(社会科学版),2022,42(1): 120-130.
- [34] 钱雨,孙新波,孙浩博,等. 数字化时代敏捷组织的构成要素、研究框架及未来展望[J]. 研究与发展管理
- [35] 徐源钰,苏旭中,刘燕卿,等. 基于 VPN 技术的竹节纱生产远程监控系统[J]. 制造业自动化,2022,44(2): 16-19.
- [36] 田苗,王军方,黄健畅,等. 唐山市柴油车远程监控综合管控平台的开发及应用[J]. 环境科学研究,2021,34(1): 132-140.
- [37] 吴双玉,陆艺,郭斌. 制造车间设备远程监控系统开发[J]. 仪表技术与传感器,2021(6): 72-76.

- [38] 王岩,高建波,张传锦,等. 生产现场远程监控方法与系统[J]. 现代制造工程,2020(1): 113-117,57.
- [39] 张斌. 矿用皮带输送机远程监控系统设计与研究[J]. 机械工程与自动化,2022(1): 192-193,196.
- [40] 杨帆,王钰涌,张沛航,等. 基于 Expert-PID 算法的矿山球磨机物联网控制系统的设计[J]. 计算机测量与控制,2022,30(3): 120-125.
- [41] 王博,肖金凤,贾磊,等. 粒子群优化模糊 PID 的塔式起重机定位和防摆研究[J]. 南华大学学报(自然科学版),2021,35(2): 47-52.
- [42] 滕琦,刘庆阁. 基于自整定模糊 PID 算法的浓缩工艺温度控制研究[J]. 长春理工大学学报(自然科学版),2021,44(2): 80-85.
- [43] 吴俊鸿,梁青,连彩云,等. 基于专家 PID 的变频空调频率调节方法研究[J]. 制冷与空调,2020,20(8): 12-15.
- [44] 邓本再,王江银,张中景,等. 基于专家 PID 控制的足球机器人截球的研究[J]. 工业控制计算机,2010,23(8): 63-64.
- [45] 孟涛,张彦,王勇. 基于 BP 神经网络 PID 的活塞式深海压力传感器压力控制研究[J]. 合肥工业大学学报(自然科学版),2022,45(1): 24-29.
- [46] 费春国,吴婷娜. 改进的神经网络 PID 在空调温度控制中的应用[J]. 中国民航大学学报,2022,40(1): 34-39.
- [47] 吕晓丹,吴次南. 改进型模糊神经网络 PID 控制器的设计与仿真[J]. 数据采集与处理,2021,36(2): 365-373.
- [48] 唐伎玲,赵宏伟,王婷婷,等. LQR 优化的 BP 神经网络 PID 控制器设计[J]. 吉林大学学报(理学版),2020,58(3): 651-658.
- [49] 张锦辉,李彦明,齐文超,等. 基于神经网络 PID 的丘陵山地拖拉机姿态同步控制系统[J]. 农业机械学报,2020,51(12): 356-366.
- [50] 陈舒平,熊光明,陈慧岩, et al. 基于 MPC 的考虑时间最优速度的高速无人驾驶车辆路径跟踪和 PID 速度控制[J]. 中南大学学报(英文版),2020,27(12): 3702-3720.
- [51] 于树友,冯阳阳,KIM JUNG-SU,等. 非线性预测控制终端约束集的优化[J]. 自动化学报,2022,48(1): 144-151.
- [52] Wang Bo,He Mengyi,Wang Xingyu,Tang Hongyu,Zhu Xianglin. A multi-model predictive control method for the Pichia pastoris fermentation process based on relative error weighting algorithm[J]. Alexandria Engineering Journal,2022,61(12).
- [53] Song Yuyan,Wang Yuhong,Zeng Qi,Zheng Zongsheng,Liao Jianquan,Liao Yiben. A Q-learning based robust MPC method for DFIG to suppress the rotor overcurrent[J]. International Journal of Electrical Power and Energy Systems,2022,141.

- [54] Brandi Silvio, Gallo Antonio, Capozzoli Alfonso. A predictive and adaptive control strategy to optimize the management of integrated energy systems in buildings[J]. Energy Reports, 2022, 8.
- [55] 史亚贝. 基于遗传 PID 的播种机排种器数控加工误差优化研究[J]. 农机化研究, 2022, 44(05): 110-113.
- [56] 刘伟, 马彪, 马利强, 陈雪辉, 俞传阳, 黄磊, 李昊. 卡尔曼滤波融合遗传 PID 控制算法在提高播种精度中的应用[J]. 安徽农业大学学报, 2021, 48(04): 674-679.
- [57] 许设, 卢炽华. 基于遗传 PID 算法的自适应巡航控制[J]. 汽车实用技术, 2020(08): 45-49+56.
- [58] Kotb Hossam, Yakout Ahmed H., Attia Mahmoud A., Turkey Rania A., AboRas Kareem M.. Speed control and torque ripple minimization of SRM using local unimodal sampling and spotted hyena algorithms based cascaded PID controller[J]. Ain Shams Engineering Journal, 2022, 13(4).
- [59] Nayak Jyoti Ranjan, Shaw Binod, Sahu Binod Kumar, Naidu Karanam Appala. Application of optimized adaptive crow search algorithm based two degree of freedom optimal fuzzy PID controller for AGC system[J]. Engineering Science and Technology, an International Journal, 2022, 32.
- [60] 禹鑫焱, 赵嘉楠, 应皓哲, 欧林林, 冯远静. 基于 FreeRTOS 的嵌入式智能骑行台[J]. 电子技术应用, 2022, 48(02): 84-90.
- [61] 李华辉, 肖云波, 沈勇, 邓斌. 基于 FreeRTOS 和 Speex 编解码器的语音处理系统设计[J]. 单片机与嵌入式系统应用, 2022, 22(02): 81-84+87.
- [62] 杨建华, 李正, 赵好, 王少文. 基于肌电信号的嵌入式手势识别系统设计[J]. 自动化与仪表, 2021, 36(12): 62-66.
- [63] 靳大为, 何磊. FreeRTOS 在液压支架电液控系统中的应用[J]. 现代工业经济和信息化, 2021, 11(09): 126-128.
- [64] 琚子晗, 白贺, 杨喜童. 基于 FreeRTOS 与 ARM 的智能探索机器人系统设计与实现[J]. 机械工程师, 2021(06): 37-39+42.
- [65] 熊一鹏, 岳伟. FreeRTOS 多任务调度机制在监控单元中的应用[J]. 单片机与嵌入式系统应用, 2021, 21(09): 64-66.
- [66] Oliveira Pedro M., Palma Jonathan M., Lacerda Márcio J.. [formula omitted] state-feedback control for discrete-time cyber-physical uncertain systems under DoS attacks[J]. Applied Mathematics and Computation, 2022, 425.
- [67] Shi Banban, Mao Xuerong, Wu Fuke. Stabilisation of hybrid system with different structures by feedback control based on discrete-time state observations[J]. Nonlinear Analysis : Hybrid Systems, 2022, 45.

- [68] Xue Yuntao,Luo Xin,Xie Jianxiang,Ma Daoyuan,Li Mingxian. Influence and comparison of P/Q-control based VSC-HVDC system on the grid power oscillation damping[J]. Energy Reports,2022,8(S5).
- [69] 徐晨,王鹿军,郭炅,吴铁洲,申喜.双有源桥-超级电容储能系统状态反馈模型预测控制策略[J].现代电子技术,2022,45(08): 137-142.
- [70] 夏国清,孙显信,任哲达.基于状态反馈控制器的多无人水面船集群控制[J/OL].控制与决策: 1-7[2022-04-13].
- [71] 寇发荣,杨慧杰,张新乾,等.采用状态反馈的无人车路径跟踪横向控制[J].机械科学与技术,2022,41(1): 143-150.
- [72] 孙洁茹,陈晓宁,王健,等.基于 Qt 的探测器温控上位机软件设计[J].安徽大学学报(自然科学版),2022,46(1): 61-67.
- [73] 贾贝贝,康明才.基于 QT 的嵌入式系统文件传输上位机软件设计[J].电子设计工程,2022,30(3): 122-125,130. DOI: 10.14022/j.issn1674-6236.2022.03.027.
- [74] 许梦华.基于 Qt 的地面模拟飞行控制系统软件设计与实现[J].电子测试,2022(1): 29-31,101. DOI: 10.3969/j.issn.1000-8519.2022.01.007.
- [75] 侯彭亮,郭苹,王展鹏,等.基于 Qt 的雷达显示及手机客户端设计[J].电子技术应用,2021,47(8): 128-132.
- [76] 乔华.基于 QT 的炮口冲击波测试系统软件设计[J].中北大学学报(自然科学版),2021,42(2): 140-145.
- [77] 曾明辉,谈宏华,邢栢豪,等.基于 QT 和智能网关的智能家居系统设计[J].自动化与仪表,2021,36(10): 28-32.
- [78] Verimatrix to collaborate with Qt Company at Embedded World 2020 to showcase IoT shielding[J]. Worldwide Computer Products News,2020.

附录1

死锁检测组件程序

```
struct source_type {
    uint64 id;
    enum Type type;
    uint64 lock_id;
    int degress;
};
struct vertex {
    struct source_type s;
    struct vertex *next;
};
struct task_graph {
    struct vertex list[MAX];
    int num;
    struct source_type locklist[MAX];
    int lockidx;
    pthread_mutex_t mutex;
};
struct task_graph *tg = NULL;
int path[MAX+1];
int visited[MAX];
int k = 0;
int deadlock = 0;
struct vertex *create_vertex(struct source_type type) {
    struct vertex *tex = (struct vertex *)malloc(sizeof(struct vertex ));
    tex->s = type;
    tex->next = NULL;
    return tex;
}
int search_vertex(struct source_type type) {
    int i = 0;
    for (i = 0; i < tg->num; i++) {
        if (tg->list[i].s.type == type.type && tg->list[i].s.id == type.id) {
            return i;
        }
    }
    return -1;
}
void add_vertex(struct source_type type) {
    if (search_vertex(type) == -1) {
        tg->list[tg->num].s = type;
        tg->list[tg->num].next = NULL;
        tg->num++;
    }
}
int add_edge(struct source_type from, struct source_type to) {
    add_vertex(from);
    add_vertex(to);
    struct vertex *v = &(tg->list[search_vertex(from)]);
    while (v->next != NULL) {
        v = v->next;
    }
    v->next = create_vertex(to);
}
int verify_edge(struct source_type i, struct source_type j) {
```

```

    if (tg->num == 0) return 0;
    int idx = search_vertex(i);
    if (idx == -1) {
        return 0;
    }
    struct vertex *v = &(tg->list[idx]);
    while (v != NULL) {
        if (v->s.id == j.id) return 1;
        v = v->next;
    }
    return 0;
}

int remove_edge(struct source_type from, struct source_type to) {
    int idxi = search_vertex(from);
    int idxj = search_vertex(to);
    if (idxi != -1 && idxj != -1) {
        struct vertex *v = &tg->list[idxi];
        struct vertex *remove;
        while (v->next != NULL) {
            if (v->next->s.id == to.id) {
                remove = v->next;
                v->next = v->next->next;
                free(remove);
                break;
            }
            v = v->next;
        }
    }
}

void print_deadlock(void) {
    int i = 0;
    printf("deadlock : ");
    for (i = 0; i < k-1; i++) {
        printf("%ld --> ", tg->list[path[i]].s.id);
    }
    printf("%ld\n", tg->list[path[i]].s.id);
}

int DFS(int idx) {
    struct vertex *ver = &tg->list[idx];
    if (visited[idx] == 1) {
        path[k++] = idx;
        print_deadlock();
        deadlock = 1;
        return 0;
    }
    visited[idx] = 1;
    path[k++] = idx;
    while (ver->next != NULL) {
        DFS(search_vertex(ver->next->s));
        k--;
        ver = ver->next;
    }
    return 1;
}

int search_for_cycle(int idx) {
    struct vertex *ver = &tg->list[idx];
    visited[idx] = 1;
    k = 0;

```

```

    path[k++] = idx;
    while (ver->next != NULL) {
        int i = 0;
        for (i = 0; i < tg->num; i++) {
            if (i == idx) continue;
            visited[i] = 0;
        }
        for (i = 1; i <= MAX; i++) {
            path[i] = -1;
        }
        k = 1;
        DFS(search_vertex(ver->next->s));
        ver = ver->next;
    }
}

void check_dead_lock(void) {
    int i = 0;
    deadlock = 0;
    for (i = 0; i < tg->num; i++) {
        if (deadlock == 1) break;
        search_for_cycle(i);
    }
    if (deadlock == 0) {
        printf("no deadlock\n");
    }
}

static void *thread_routine(void *args) {
    while (1) {
        sleep(5);
        check_dead_lock();
    }
}

void start_check(void) {
    tg = (struct task_graph*)malloc(sizeof(struct task_graph));
    tg->num = 0;
    tg->lockidx = 0;
    pthread_t tid;
    pthread_create(&tid, NULL, thread_routine, NULL);
}

int search_lock(uint64 lock) {
    int i = 0;
    for (i = 0; i < tg->lockidx; i++) {
        if (tg->locklist[i].lock_id == lock) {
            return i;
        }
    }
    return -1;
}

int search_empty_lock(uint64 lock) {
    int i = 0;
    for (i = 0; i < tg->lockidx; i++) {
        if (tg->locklist[i].lock_id == 0) {
            return i;
        }
    }
}

```

```

    }
    return tg->lockidx;
}

int inc(int *value, int add) {
    int old;
    __asm__ volatile(
        "lock;xaddl %2, %1;"
        : "=a"(old)
        : "m"(*value), "a" (add)
        : "cc", "memory"
    )
    return old;
}

void print_locklist(void) {
    int i = 0;
    for (i = 0; i < tg->lockidx; i++) {
        printf("threadid      :      %ld,      lockid      :      %ld\n",      tg->locklist[i].id,      tg->locklist[i].lock_id);
    }
}

void lock_before(uint64 thread_id, uint64 lockaddr) {
    int idx = 0;
    for(idx = 0; idx < tg->lockidx; idx++) {
        if ((tg->locklist[idx].lock_id == lockaddr)) {
            struct source_type from;
            from.id = thread_id;
            from.type = PROCESS;
            add_vertex(from);
            struct source_type to;
            to.id = tg->locklist[idx].id;
            tg->locklist[idx].degress++;
            to.type = PROCESS;
            add_vertex(to);
            if (!verify_edge(from, to)) {
                add_edge(from, to);
            }
        }
    }
}

void lock_after(uint64 thread_id, uint64 lockaddr) {
    int idx = 0;
    if (-1 == (idx = search_lock(lockaddr))) {
        int eid = search_empty_lock(lockaddr);
        tg->locklist[eid].id = thread_id;
        tg->locklist[eid].lock_id = lockaddr;
        inc(&tg->lockidx, 1);
    } else {
        struct source_type from;
        from.id = thread_id;
        from.type = PROCESS;
        struct source_type to;
        to.id = tg->locklist[idx].id;
        tg->locklist[idx].degress--;
        to.type = PROCESS;
        if (verify_edge(from, to))
    }
}

```

```

        remove_edge(from, to);
        tg->locklist[idx].id = thread_id;
    }
}

void unlock_after(uint64 thread_id, uint64 lockaddr) {
    int idx = search_lock(lockaddr);
    if (tg->locklist[idx].degress == 0) {
        tg->locklist[idx].id = 0;
        tg->locklist[idx].lock_id = 0;
    }
}

int pthread_mutex_lock(pthread_mutex_t *mutex) {
    pthread_t selfid = pthread_self();
    lock_before(selfid, (uint64)mutex);
    pthread_mutex_lock_f(mutex);
    lock_after(selfid, (uint64)mutex);
}

int pthread_mutex_unlock(pthread_mutex_t *mutex) {
    pthread_t selfid = pthread_self();
    pthread_mutex_unlock_f(mutex);
    unlock_after(selfid, (uint64)mutex);
}

```

附录2

远程监控程序

```
void monitor () {
    while (1) {
        printf("等待控制端发起连接...\n");
        clientSock = accept(serverSocket, &client, &nSize);
        printf("控制端已经接入! \n");
        cap.open(0);
        while (1) {
            // 接受“监控指令”
            char buff[4096];
            printf("等待指令...\n");
            int ret = recv(clientSock, buff, sizeof(buff), 0);
            if (ret <= 0) break;

            // 判断收到的是不是“监控”指令
            if (strcmp(buff, "JIAN_KONG") == 0) {
                // 正式监控
                paiZhao("jk.jpg");
                printf("已经拍摄! \n");
                FILE* file = fopen("jk.jpg", "rb");
                fseek(file, 0, SEEK_END);
                int len = ftell(file);
                send(clientSock, (char*)&len, 4, NULL);
                fseek(file, 0, SEEK_SET);
                while (1) {
                    int ret = fread(buff, 1, 4096, file);
                    if (ret <= 0) break;
                    send(clientSock, buff, ret, NULL);
                }
                fclose(file);
                printf("已经发送完毕\n");
                send(clientSock, pack, 4, NULL);
                printf("结束标记已经发送完毕\n");
            }
        }
        closesocket(clientSock);
        cap.release();    //关闭摄像头
    }
}

void receivePhoto() {
    int len;
    // 先接受照片的长度
    recv(serverSocket, (char*)&len, 4, NULL);
    char buff[4096];
    FILE* file = fopen("jk.jpg", "wb");
    int count = 0;
    while (1) {
        int ret = recv(serverSocket, buff, sizeof(buff), NULL);
        if (ret < 0) {
            printf("接受失败!\n");
            break;
        } else if (ret == 0) {
            printf("对端关闭\n");
            break;
        }
    }
}
```

```

    }
    else {
        printf("接收到 %d 字节\n", ret);
        // 把接收到的数据，保存到文件里
        fwrite(buff, ret, 1, file);
    }
    count = count + ret;
    if (count == len) {
        break;
    }
}
fclose(file);
loadimage(0, _T("jk.jpg"));
}

```

作者简历

一、基本情况

姓名：邓凯旋 性别：男 民族：汉 出生年月：1995-07-19 籍贯：江苏徐州

2014-09—2018-07 华北科技学院电子信息工程学院学士

2019-09—2022-07 华北科技学院安全工程学院硕士

二、学术论文

1. 邓凯旋. C语言实现推箱子[J]. 电脑编程技巧与维护, 2021(6): 68-70.
2. 邓凯旋. Qt实现考试系统[J]. 电脑编程技巧与维护, 2021(8): 16-18, 23.

三、研究项目

1. 基于 STM32 的压滤机控制器的研究与开发. 中央高校科研基金, 项目编号: 3142019038, 项目参与人员

学位论文原创性声明

本人郑重声明：所呈交的学位论文《 》，是本人在导师指导下，在华北科技学院攻读学位期间进行的研究工作所取得的成果。据我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名:

年 月 日

学位论文数据集

关键词*	密级*	中图分类号*	UDC	论文资助
智能控制；模糊控制；PID；STM32	公开	TP273		
学位授予单位名称*	学位授予单位代码*	学位类别*	学位级别*	
华北科技学院	11104	工学	硕士	
论文题名*		并列题名*		论文语种*
基于STM32的智能控制器的研究与开发		Research and Development of Intelligent Controller Based on STM32		中文
作者姓名*	邓凯旋	学号*	201908522454	
培养单位名称*	培养单位代码*	培养单位地址	邮编	
华北科技学院	11104	河北三河燕郊	065201	
学科专业*	研究方向*	学制*	学位授予年*	
安全工程	安全生产自动化与信息化	三年	2022	
论文提交日期*		2022年6月		
导师姓名*	翟延忠	职称*	教授	
评阅人		答辩委员会主席*	答辩委员会成员	
		洪利	蔡建美、张全柱、张涛、黎冠	
电子版论文提交格式 文本 (<input checked="" type="checkbox"/>) 图像 (<input type="checkbox"/>) 视频 (<input type="checkbox"/>) 音频 (<input type="checkbox"/>) 多媒体 (<input type="checkbox"/>) 其他 (<input type="checkbox"/>) 推荐格式: application/msword; application/pdf				
电子版论文出版 (发布) 者	电子版论文出版 (发布) 地		权限声明	
论文总页数				
注：共33项，其中带*为必填数据，共22项。				

