本科生毕业设计

智能手持式激光测距仪

姓名	王恺
学号	20201804070 质量技术监督学院
院系	测控技术与仪器
专业	刘琨
指导教师	

第 1 章. 摘要

本课程设计项目设计了一款基于 LS-K-100 激光测距模块和 ESP32 的高精度激光测距仪。该测距仪设计用于实现 100 米范围内的 ±2mm 高精度测量，适用于家装测量、工业控制等领域。报告从硬件选型、原理图 PCB 设计、软件编程到结构建模等方面，全面阐述了产品开发流程，并提供了详细的开发步骤和实现细节。通过本课程设计掌握硬件设计与软件编程的基础知识，提升解决实际问题的能力。未来工作可以进一步优化系统性能，如提高测量精度、降低功耗、增加无线通信功能等。

关键词**:** 激光测距；PLS-K-100；ESP32；高精度测量

目录

第 2 章. 前言

在现代工业和日常生活中，精确测量成为了提高工作效率和质量的关键因素。激光测距仪作为一种高精度、非接触式的测量工具，其在建筑测量、室内装修、机械制造等领域的应用日益广泛。随着技术的发展，对激光测距仪的测量精度和操作便捷性提出了更高的要求。

本课程设计旨在开发一款基于 PLS-K-100 激光测距模块和 ESP32 微控制器的高精度激光测距仪。PLS-K-100 以其高精度、快速响应和易于集成的特点，成为本项目的核心元器件。而 ESP32 作为多功能的微控制器，不仅提供了强大的处理能力，还支持 Wi-Fi 和蓝牙通信，为产品的智能化和网络化提供了可能。

设计目标是实现一款能够在 100 米范围内提供 ±2mm 测量精度的激光测距仪，同时具备用户友好的操作界面和低功耗设计。本项目采用开源硬件和软件平台，以便于快速开发和迭代，同时也便于理解和学习整个设计流程。

通过本课程设计，深入了解硬件电路设计、微控制器编程、以及产品原型制作等关键技术。此外，也将学习到如何将理论知识应用于实际项目中，解决实际问题，培养创新思维和工程实践能力。最终，本项目的成功实施将为学生提供一个完整的电子产品开发经验，为未来的职业生涯打下坚实的基础。

第 3 章. 设计目标与要求

本课程设计的核心目标是开发一款高精度激光测距仪，以下为项目设计的技术要求和性能指标：

1. 测量范围： 能够在 100 米范围内进行测量，以适应多种测量场景，包括室内和室外环境。
2. 测量精度： 测量精度 ±2mm，以确保测量结果的准确性，满足高精度测量的需求。
3. 响应时间： 具有快速响应能力，测量响应时间再 1 秒左右，以便于用户实时获取测量数据。
4. 电源： 设备采用单节锂电池供电，以便于携带和使用，同时考虑到电池的续航能力和充电便捷性。
5. 显示： 配备 OLED 屏幕，用于直观显示测量结果和设备状态，提高用户体验。
6. 操作： 设备通过实体按键操作功能，包括开关机、测量模式切换、单位选择等，确保用户可以在各种条件下操作实现实现需求。
7. 环境适应性：具有良好的环境适应性，能够在 −30℃ 至 +60℃ 的温度范围内正常工作，满足不同气候条件下的使用需求。
8. 安全性： 符合相关安全标准，确保在使用过程中不会对用户或周围环境造成危害。
9. 易用性： 设计应简洁明了，易于用户理解和操作，同时考虑设备的便携性和耐用性。
10. 可扩展性： 设计留有有良好的可扩展性，便于未来根据需要增加新的功能或进行技术升级。

通过实现上述目标，本课程设计旨在提供一个高性能、易用、可靠的激光测距仪解决方案，不仅能够满足当前的测量需求，还具有一定的前瞻性和发展潜力。

第 4 章. 系统设计

4.1. 硬件设计

4.1.1. 硬件设计概述

硬件设计是实现高精度激光测距仪功能的基础，涉及元器件的选择、电路设计、PCB 布局以及外壳设计等多个方面。本节将详细介绍硬件设计的各个环节。

4.1.2. 核心元器件选择

核心元器件的选择对整个系统的性能至关重要。本项目的核心元器件为 PLS-K-100 激光测距模块，由它提供高精度的测量数据。

PLS-K-100 激光测距模块：

PLS-K-100 是一款高精度、长距离的激光测距模块，它采用了先进的激光测距技术，能够在 100 米范围内提供 ±2mm 的测量精度。该模块具有以下特点：

- 高精度：测量精度高达 ±2mm，适用于需要精确测量的应用场景。
- 长测量距离：最大测量距离可达 100 米，适用于室内外各种测量任务。
- 快速响应：测量速度快，能够在短时间内提供测量结果。
- 串口通信：通过串口与微控制器通信，便于数据传输和处理。
- 低功耗：设计考虑了节能，适合长时间使用。
- 易于集成：模块化设计，方便与其他硬件系统集成。

4.1.3. 微控制器选择与编程

微控制器的选择基于其处理能力、通信接口以及对外部库的支持。ESP32 因其强大的处理能力、内置 Wi-Fi 和蓝牙功能，以及对 Arduino 编程环境的良好支持而被选为 MCU。编程环境采用 Arduino IDE，利用 Arduino 框架丰富的库和社区支持简化开发过程。

ESP32 是一款功能强大的 MCU 单元，它集成了 Wi-Fi、蓝牙和多个串口通信接口，非常适合于物联网(IoT) 应用。以下是 ESP32 的一些关键特性：

- 多核处理器：具有双核处理器，可以同时处理多个任务，提高系统性能。

- 丰富的通信接口：支持 Wi-Fi、蓝牙、SPI、I2C、UART 等多种通信协议，便于与其他设备连接。
- 大容量内存：具有足够的 RAM 和 ROM，可以运行复杂的应用程序以满足项目要求。
- 低功耗模式：支持多种低功耗模式，适合电池供电的便携设备。
- 开发环境友好：支持 Arduino 开发框架，有丰富的库和社区支持，易于开发和调试。

选择 PLS-K-100 激光测距模块作为测量核心，是因为其高精度和长距离测量能力，而选择 ESP32 微控制器则是因为它强大的处理能力和丰富的通信接口，能够满足系统对数据处理和用户交互的需求。这两个核心元器件的结合，使得我们的激光测距仪能够实现高精度、快速响应和用户友好的操作体验。

4.1.4. 显示模块设计

显示模块采用 OLED 屏幕，优点是低功耗、高对比度和优秀的可编程性。OLED 屏幕通过 I2C 接口与 ESP32 通信，显示测量结果和系统状态。

4.1.5. 输入模块设计

输入模块包括一组按键，用于用户操作。按键设计考虑了耐用性以及不同环境和复杂使用场景下的操作便利性，确保用户可以轻松地进行测量、切换模式等操作。

4.1.6. 电源管理设计

电源管理模块包括电池、充电芯片和降压芯片。电池选用单节锂电池，以便于携带和更换。TP5100 充电芯片用于电池充电，LP3220 降压芯片则用于将电池电压稳定地降压至 3.3V。

4.1.7. 外壳设计

外壳设计暂时采用亚克力板与铜柱螺栓直架固定。提供简单的机械支撑和隔离功能，结构稳固可靠且易于安装，同时能够满足机械和隔离的要求，保护 PCB 板并提供额外支撑。

4.1.8. PCB 设计

PCB 设计是硬件设计的另一个关键环节。设计时需考虑元器件的布局、走线以及散热。PCB 布局应合理，以便于焊接和维修。走线应尽量短且避免交叉，以减少干扰。PCB 设计还应考虑散热问题，确保设备在长时间工作下不会过热。

4.1.9. 焊接与组装

焊接是将PCB上的元器件固定并连接起来的过程。焊接质量直接影响到设备的性能和可靠性。焊接顺序和方法对焊接质量至关重要。组装过程包括将焊接好的PCB装入外壳，安装显示屏和按键，以及连接电池。

4.1.10. 测试与调试

硬件组装完成后，需要进行全面的测试和调试，包括电源测试、功能测试、性能测试等，确保设备按照设计要求正常工作。

4.1.10.1. 性能优化

在测试过程中，可能会发现一些性能瓶颈或不足之处。针对这些问题，可以通过优化电路设计、改进焊接工艺或调整软件算法等方式进行性能优化。

4.1.11. 结论

硬件设计是一个复杂的过程，需要综合考虑多种因素。通过精心的设计和严谨的测试，可以确保激光测距仪的高性能和可靠性。本节详细介绍了硬件设计的各个环节，为后续的软件设计和系统实现奠定了坚实的基础。

4.2. 软件设计

软件设计是实现激光测距仪功能的关键部分，它涉及到程序的逻辑结构、用户界面(UI)设计以及与硬件的交互。本节将详细介绍软件设计的各个方面。

4.2.1. 编程环境与语言

本项目采用Arduino编程环境，使用C++语言进行开发。Arduino是一个开源的电子原型平台，它包含了硬件（各种型号的Arduino板）和软件（Arduino IDE）。Arduino IDE提供了一个易于使用的编程环境，支持C++语言，并且有丰富的库和社区支持。

4.2.2. 程序结构

程序的主要结构包括初始化、主循环、按键扫描、数据处理和显示更新等部分。程序在初始化阶段设置硬件引脚、初始化显示屏和串口通信等。主循环中不断检测按键状态，根据按键输入执行相应的操作，如测量距离、切换测量模式、调整单位等。数据处理部分负

责解析激光测距仪返回的数据，并计算出距离。显示更新部分则将测量结果和系统状态显示在 OLED 屏幕上。

4.2.3. 功能实现

软件的主要功能包括：

- 单次测量：用户按下测量键，系统发送测量指令，等待激光测距仪返回数据，然后显示测量结果。
- 连续测量：用户进入连续测量模式，系统持续发送测量指令，实时更新显示测量结果。
- 激光开关：用户可以通过按键控制激光的开启和关闭。
- 单位切换：用户可以切换测量单位，如毫米、厘米和米。
- 电池电量检测：系统可以检测电池电量，并在屏幕上显示。
- 蜂鸣器控制：用户可以通过按键控制蜂鸣器的开启和关闭。

4.2.4. OLED 显示驱动

使用了 U8g2 库来驱动 OLED 显示屏，以及可能的其他第三方库来处理串口通信、ADC 转换等任务。这些库提供了现成的功能，使得开发者可以专注于实现产品的核心功能。

手动写入可能用到的中文字符集，定义文字显示像素。

```
++

U8G2_SH1106_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, /* reset=*/ U8X8_PIN_NONE);//1.30
寸oled屏幕参数(SH1106)

//U8G2_SSD1306_128X64_NONAME_F_SW_I2C u8g2(U8G2_R0, /* clock=*/ SCL, /* data=*/
SDA, /* reset=*/ U8X8_PIN_NONE);//0.96寸oled屏幕参数(SSD1306)


/*距(0) 离(1) 单(2) 位(3) 切(4) 换(5) 错(6) 误(7)
  码(8) 激(9) 光(10) 头(11) 开(12) 关(13) 整(14) 机(15)
  硬(16) 件(17) 故(18) 障(19) 无(20) 效(21) 信(22) 号(23)
  不(24) 稳(25) 定(26) 太(27) 弱(28) 强(29) 背(30) 景(31)
  光(32) 无(33) 效(34) 的(35) 测(36) 量(37) 结(38) 果(39)
```

```
  目(40) 标(41) 超(42) 出(43) 射(44) 程(45) 电(46) 池(47)
  可(48) 信(49) 度(50)
*/

static const unsigned char PROGMEM str0[] = {0x00, 0x00, 0xBE, 0x7F, 0xA2, 0x00,
 0xA2, 0x00, 0xA2, 0x00, 0xBE, 0x3F, 0x88, 0x20, 0x88, 0x20, 0xBA, 0x20, 0x8A,
0x20, 0x8A, 0x3F, 0x8A, 0x00, 0xBA, 0x00, 0x87, 0x00, 0x80, 0x7F, 0x00, 0x00}; /
*"距",0*/
static const unsigned char PROGMEM str1[] = {0x40, 0x00, 0x80, 0x00, 0xFF, 0x7F,
0x00, 0x00, 0x28, 0x0A, 0xC8, 0x09, 0x28, 0x0A, 0xF8, 0x0F, 0x80, 0x00, 0xFE, 0x3F,
0x42, 0x20, 0x22, 0x22, 0xF2, 0x27, 0x22, 0x24, 0x02, 0x28, 0x02, 0x10}; /*"离",
1*/
…
…
static const unsigned char PROGMEM str54[] = {0x00, 0x20, 0x00, 0x70, 0x00, 0xF8,
0x00, 0xA8, 0x00, 0x20, 0x00, 0x00, 0x00, 0x20, 0xFF, 0x21, 0x01, 0x21, 0x7D, 0x01,
 0x45, 0x21, 0x45, 0x21, 0x7D, 0x21, 0x01, 0x01, 0x01, 0x21, 0x01, 0x21, 0x01,
 0x21, 0x01, 0x21, 0x01, 0x01, 0x01, 0x21, 0x01, 0x21, 0x01, 0x21, 0x01, 0x01,
 0x01, 0x21, 0x01, 0x21, 0x01, 0x21, 0x01, 0x01, 0x01, 0x21, 0x01, 0x21, 0x01,
0x21, 0x01, 0x01, 0xFF, 0xF9}; /*"后基准",54*/
static const unsigned char PROGMEM str55[] = {0x00, 0x20, 0x00, 0x70, 0x00, 0xF8,
0x00, 0xA8, 0x00, 0x20, 0x00, 0x00, 0x00, 0x20, 0xFF, 0xF9, 0x01, 0x01, 0x7D, 0x01,
 0x45, 0x01, 0x45, 0x01, 0x7D, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x01, 0x01, 0xFF, 0x01}; /*"前基准",55*/
```

以上代码定义了一系列的字符串数组，用于在 OLED 显示屏上显示中文字符。使用 PROGMEM 关键字，将其存储在程序存储器（而非 RAM）中，以节省内存空间和提高程序效率。

通过 str0[] 这样的数组定义了字符，其内容是一个包含多个字节的数组，每个字节代表屏幕上的一个像素行。这些字节的值是通过位操作（如 0x00、0x7F 等）来设置的，其中 0x7F 表示该行的所有像素都亮起，而 0x00 表示该行的所有像素都熄灭。

这些字符串数组可以被用来在 OLED 显示屏上显示用户界面，例如在测量结果旁边显示相关的提示信息，或者在系统状态发生变化时提供反馈。通过这种方式，用户可以直观地了解设备的工作状态和测量结果。

在 u8g2 库的函数中指定位置绘制位图，在 OLED 屏幕上显示中文字符和图标，增强用户体验。

4.2.5. 测距模块驱动

根据激光测距模块规格书，定义与激光测距模块通信的协议。

4.2.5.1. 发送指令到测距模块串口

```
++

//发送到测距模块串口


void measursend()
{
  unsigned char S_A_M[9] = {0xAA, 0x00, 0x00, 0x20, 0x00, 0x01, 0x00, 0x00, 0x21}
; //单次自动测量
  unsigned char C_A_M[9] = {0xAA, 0x00, 0x00, 0x20, 0x00, 0x01, 0x00, 0x04, 0x25}
; //连续自动测量
   unsigned char O_L[9] = {0xAA, 0x00, 0x01, 0xBE, 0x00, 0x01, 0x00, 0x01, 0xC1}
; //打开激光
   unsigned char C_L[9] = {0xAA, 0x00, 0x01, 0xBE, 0x00, 0x01, 0x00, 0x00, 0xC0}
; //关闭激光
  unsigned char EXIT[1] = {0x58}; //退出连续自动测量
```

```
  switch (MEASUR_STATUS)
  {
    case MEASUR_OFF:
      Serial.println("测量关闭"); break;
    case CLOSE_LASER:
     Serial2.write(C_L, 9); Serial.println("关闭激光"); MEASUR_STATUS = MEASUR_OFF;
break;
    case OPEN_LASER:
      Serial2.write(O_L, 9); Serial.println("打开激光");  break;
    case SINGLE_AUTO_MEASUR:
        Serial2.write(S_A_M, 9); Serial.println("单次自动测量");   MEASUR_STATUS =
MEASUR_OFF; break;
    case CONTINUOUS_AUTO_MEASUR:
      Serial2.write(C_A_M, 9); Serial.println("连续自动测量"); break;
    case EXIT_MEASUR:
       Serial2.write(EXIT, 9); Serial.println("关闭连续自动测量"); MEASUR_STATUS =
MEASUR_OFF; break;
    default: Serial.println("测量无发送");
  }
}
```

定义 measursend() 函数用于发送指令到激光测距仪模块。这个函数根据当前的测量状态（MEASUR_STATUS）来决定发送哪种类型的指令。

函数首先定义了四个数组，每个数组代表一种不同的指令：

- S_A_M：用于单次自动测量。
- C_A_M：用于连续自动测量。
- O_L：用于打开激光。

- C_L：用于关闭激光。
- EXIT：用于退出连续自动测量。

这些数组包含了发送给激光测距仪的命令字节序列，这些字节序列遵循激光测距仪的通信协议。

然后，函数使用 switch 语句根据 MEASUR_STATUS 的值来选择发送哪个数组。MEASUR_STATUS 是一个枚举类型，定义了激光测距仪的不同操作状态。

对于每种状态，函数执行以下操作：

- MEASUR_OFF：打印一条消息到串行监视器，说明测量已关闭，不发送任何指令。
- CLOSE_LASER：发送关闭激光的指令，然后重置 MEASUR_STATUS 为 MEASUR_OFF 。
- OPEN_LASER：发送打开激光的指令。
- SINGLE_AUTO_MEASUR：发送单次自动测量的指令，然后重置 MEASUR_STATUS 为 MEASUR_OFF 。
- CONTINUOUS_AUTO_MEASUR：发送连续自动测量的指令。
- EXIT_MEASUR：发送退出连续自动测量的指令，然后重置 MEASUR_STATUS 为 MEASUR_OFF。

在每种情况下，函数都会打印一条消息到串行监视器，说明发送了哪种类型的指令。

4.2.5.2. 将测距模块返回结果保存

```
++
enum
{
  MEASUR_OFF,
  CLOSE_LASER,
  OPEN_LASER,
  SINGLE_AUTO_MEASUR,
  CONTINUOUS_AUTO_MEASUR,
  EXIT_MEASUR,
} MEASUR_STATUS;
```

MEASUR_STATUS 作为一个枚举类型（enum），定义了激光测距仪测量状态的不同可能值。枚举类型是一种用户定义的数据类型，它允许将整数值与特定的名字关联起来，使得代码更加清晰和易于理解。

这个枚举定义了以下状态：

1. MEASUR_OFF：表示测量功能关闭，激光测距仪不进行任何测量操作。
2. CLOSE_LASER：表示关闭激光，即关闭激光测距仪的激光发射器。
3. OPEN_LASER：表示打开激光，即启动激光测距仪的激光发射器。
4. SINGLE_AUTO_MEASUR：表示进行单次自动测量，即用户发起一次测量请求，设备自动进行测量并返回结果。
5. CONTINUOUS_AUTO_MEASUR：表示进入连续自动测量模式，设备会持续进行测量并实时返回结果。
6. EXIT_MEASUR：表示退出连续自动测量模式，设备停止连续测量。

使用枚举类型使得代码更加直观，便于维护和调试。当需要更改某个状态的值之时，只需在枚举定义中进行修改，而不需要在代码的其他部分进行查找和替换。此外，枚举类型为状态值提供了一个明确的范围，防止了意外地使用非法的值，也有助于减少错误。

4.2.6. 蜂鸣器

在按照顺序执行法时，让蜂鸣器 IO 变高，延迟 200ms，然后变低，导致 CPU 都被蜂鸣器时间占用。避免这种情况，使用 millis 函数检测按键时间，按键时间大于 200ms ，启动蜂鸣器。

```
++

//蜂鸣器开


void buzzeron200ms()
{
  if (BUZZER_STATUS == BUZZER_ON)
  {
```

```
    digitalWrite(Buzzer, HIGH);
    buzzerstarttime = millis();//蜂鸣器开始计时
  }
  if (BUZZER_STATUS == BUZZER_OFF)
  {
    digitalWrite(Buzzer, LOW);
  }
}
//蜂鸣器复位
void buzzerreset()
{
  buzzerendtime = millis();
  if (buzzerendtime < (buzzerstarttime + Time_200ms_setting))//蜂鸣器结束计时<按下计时+200ms
  {
    digitalWrite(Buzzer, LOW);//关闭蜂鸣器
  }
}
```

4.2.7. 单位转换

定义 unitmanage() 函数实现了激光测距仪的单位转换功能。这个功能允许用户通过按下一个特定的按键（UnitButton 定义的串口所连按键）来在毫米（mm）、厘米（cm）和米（m）之间切换测量单位。

```
++
//单位切换
void unitmanage()
{
```

```
if (UnitKeyState == 1 && Unitkeyflag == 0)
{
  delay(10);
  if (UnitKeyState == 1 && UNIT_STATUS == UNIT_MM)//MM CM M切换
  {
    UNIT_STATUS = UNIT_CM;
  }
  else if (UnitKeyState == 1 && UNIT_STATUS == UNIT_CM)
  {
    UNIT_STATUS = UNIT_M;
  }
  else if (UnitKeyState == 1 && UNIT_STATUS == UNIT_M)
  {
    UNIT_STATUS = UNIT_MM;
  }
  Unitkeyflag = 1;
  buzzeron200ms();
}
else if (UnitKeyState == 0)
{
  delay(10);
  if (UnitKeyState == 0)
  {
    Unitkeyflag = 0;
  }
```

```
  }
}
```

函数首先检查 UnitKeyState 是否为 1，这表示单位转换按钮被按下。同时，它还检查 Unitkeyflag 是否为 0，以确保这个操作不是重复的。如果这两个条件都满足，那么程序会进入一个 if-else 结构，根据当前的 UNIT_STATUS 值来决定新的单位。

- 如果当前单位是毫米（UNIT_STATUS == UNIT_MM），则新的单位设置为厘米（UNIT_STATUS = UNIT_CM）。
- 如果当前单位是厘米（UNIT_STATUS == UNIT_CM），则新的单位设置为米（UNIT_STATUS = UNIT_M）。
- 如果当前单位是米（UNIT_STATUS == UNIT_M），则新的单位设置为毫米（UNIT_STATUS = UNIT_MM）。

每次单位改变后，Unitkeyflag 被设置为 1，这表示单位已经改变，直到下一次按键操作。如果在单位改变后 UnitKeyState 变为 0，Unitkeyflag 也会被重置为 0，以准备下一次的单位切换操作。

函数调用 buzzeron200ms() 来激活蜂鸣器 200 毫秒，为用户提供操作反馈，发出声音，让用户知道单位已经成功切换。

4.2.8. 基准切换

```
//基准开关
void basemanage()
{
  if (BaseKeyState == 1 && Basekeyflag == 0)
  {
    delay(10);
    if (BaseKeyState == 1 && BASE_STATUS == BASE_FRONT)//基准切换
    {
      BASE_STATUS = BASE_BACK;
```

```
    }
    else if (BaseKeyState == 1 && BASE_STATUS == BASE_BACK)
    {
      BASE_STATUS = BASE_FRONT;
    }
    Basekeyflag = 1;
    buzzeron200ms();
  }
  else if (BaseKeyState == 0)
  {
    delay(10);
    if (BaseKeyState == 0)
    {
      Basekeyflag = 0;
    }
  }
}
```

basemanage()函数用于处理基准切换的按键操作,基准切换按钮的状态,并根据当前的基准状态来切换基准位置。

函数首先检查 BaseKeyState 是否为 1，这表示基准切换按钮被按下。同时，它还检查 Basekeyflag 是否为 0，以确保这个操作不是重复的。如果这两个条件都满足，那么程序会进入一个 if-else 结构，根据当前的 BASE_STATUS 值来决定新的基准状态。

- 如果当前基准状态是 BASE_FRONT（基准在前），则新的基准状态设置为 BASE_BACK（基准在后）。
- 如果当前基准状态是 BASE_BACK（基准在后），则新的基准状态设置为 BASE_FRONT（基准在前）。

每次基准状态改变后，Basekeyflag 被设置为 1，这表示基准已经改变，直到下一次按键操作。如果在基准状态改变后 BaseKeyState 变为 0，Basekeyflag 也会被重置为 0，以准备下一次的基准切换操作。

4.2.9. 2.2.9 错误处理

当程序执行错误时，提示错误码，帮助用户在使用时排除错误。

```
++
{
        Serial.println("错误码校验和相等，错误码有效");
        switch (Distance_raw[7])
        {
          case 0x00: Serial.println("错误码 00:无错误"); errorcode = 1; break;
        case 0x01: Serial.println("错误码 01:输入功率过低，功率电压应>= 2.2V"); errorcode
= 2; break;
         case 0x02: Serial.println("错误码 02:内在错误，没关系"); errorcode = 3; break;
          case 0x03: Serial.println("错误码 03:模块温度过低(< -20℃)"); errorcode = 4;
break;
           case 0x04: Serial.println("错误码 04:模块温度过高(> + 40℃)"); errorcode =
5; break;
          case 0x05: Serial.println("错误码 05:目标超出射程"); errorcode = 6; break;
         case 0x06: Serial.println("错误码 06:无效的测量结果"); errorcode = 7; break;
          case 0x07: Serial.println("错误码 07:背景光太强"); errorcode = 8; break;
          case 0x08: Serial.println("错误码 08:激光信号太弱"); errorcode = 9; break;
         case 0x09: Serial.println("错误码 09:激光信号太强"); errorcode = 10; break;
          case 0x0A: Serial.println("错误码 0A:硬件故障 1"); errorcode = 11; break;
          case 0x0B: Serial.println("错误码 0B:硬件故障 2"); errorcode = 12; break;
          case 0x0C: Serial.println("错误码 0C:硬件故障 3"); errorcode = 13; break;
```

```
        case 0x0D: Serial.println("错误码 0D:硬件故障 4"); errorcode = 14; break;
        case 0x0E: Serial.println("错误码 0E:硬件故障 5"); errorcode = 15; break;
       case 0x0F: Serial.println("错误码 0F:激光信号不稳定"); errorcode = 16; break;
        case 0x10: Serial.println("错误码 10:硬件故障 6"); errorcode = 17; break;
        case 0x11: Serial.println("错误码 11:硬件故障 7"); errorcode = 18; break;
        case 0x81: Serial.println("错误码 81:无效"); errorcode = 19; break;
        default: Serial.println("错误码未知");
      }
    }
```

由于项目时间限制，错误码验证与输出部分未完成，留待后续完善。

4.2.10. 结论

软件设计是实现激光测距仪功能的关键部分。通过合理的程序结构、直观的用户界面和完善的功能实现，可以确保产品具有良好的用户体验和稳定的性能。同时，通过性能优化和错误处理，可以提高程序的可靠性和健壮性。

第 5 章. 系统实现

5.1. 硬件搭建与测试

- 使用杜邦线和面包板搭建初步硬件平台，进行功能验证和调试，确保电路设计的正确性。
- PCB 设计和焊接，确保电路连接正确无误，并通过焊接质量检查。

硬件搭建是将设计转化为实际可工作的系统的过程。这一部分将详细描述硬件搭建的步骤、测试方法以及可能出现的问题和解决方案。

5.1.1. 搭建步骤

1. 元器件准备：根据 BOM 清单准备所有所需的元器件，包括 PLS-K-100 激光测距模块、ESP32 微控制器、OLED 显示屏、按键、电源管理芯片等。
2. PCB 焊接：将元器件焊接到 PCB 板上。首先焊接较为复杂的部件，如 QFN 封装的 TP5100 芯片，然后是其他贴片元件，最后焊接插件元件。
3. 电源管理：确保电源管理电路正确连接，包括锂电池、充电芯片 TP5100、降压芯片 LP3220 等，以确保系统稳定供电。
4. 显示屏连接：将 OLED 显示屏通过 I2C 接口连接到 ESP32 ，确保显示模块正确安装并固定。
5. 按键安装：安装按键，确保按键与 PCB 板上的焊盘正确连接，并且按键操作灵活。
6. 激光测距模块连接：将 PLS-K-100 激光测距模块通过串口连接到 ESP32。
7. 组装外壳：将焊接好的 PCB 板安装到 3D 打印的外壳中，确保所有连接线和接口都正确无误。
8. 初步测试：在组装完成后，进行初步的电源测试，确保电源供应正常，无短路现象。

5.1.2. 测试方法

1. 电源测试：使用万用表检查电源输出是否符合预期，确保电源管理电路工作正常。
2. 串口通信测试：使用串口通信工具检查 ESP32 与激光测距模块之间的通信是否正常。
3. 按键功能测试：通过按下各个按键，观察系统的反应，确保按键功能正确。

4. 显示屏测试：检查 OLED 显示屏是否能够正常显示，包括文字、图标等。

5. 激光测距测试：进行实际的测距操作，验证测量结果的准确性和稳定性。

6. 系统功能测试：测试所有系统功能，包括单次测量、连续测量、激光开关控制、单位切换等。

5.1.3. 问题与解决方案

在搭建过程中可能会遇到以下问题：

- 焊接问题：如果焊接不良，可能导致电路短路或断路。解决方案是通过视觉检查和使用万用表进行检测，必要时重新焊接。
- 连接问题：电路连接错误可能导致系统无法正常工作。解决方案是仔细检查原理图和PCB布局，确保所有连接正确无误。
- 电源问题：电源供应不稳定可能导致系统工作异常。解决方案是检查电源管理电路，确保电池和充电电路工作正常。
- 通信问题：串口通信故障可能导致数据传输错误。解决方案是检查通信线路和通信协议设置。
- 测量精度问题：如果测量结果不准确，可能是激光测距模块或软件算法的问题。解决方案是检查激光测距模块的设置和软件中的测量算法。

5.1.4. 结论

硬件搭建与测试是确保系统能够正常工作的重要步骤。通过细致的搭建和全面的测试，可以发现并解决潜在的问题，确保激光测距仪的性能和可靠性。

5.2. 软件编程与调试

- 编写控制程序，实现与激光测距仪的通信，确保数据的准确传输。
- 实现按键扫描和处理逻辑，确保用户操作的准确性和系统的响应速度。
- 测试软件功能，确保测量结果的准确性和系统的稳定性，通过多次测试优化软件性能。

5.2.1. 编程环境配置

首先，需要配置 Arduino 开发环境。这包括安装 Arduino IDE，以及可能需要的其他插件，如 U8g2 库的安装，用于驱动 OLED 显示屏。确保所有必要的库和工具都已正确安装并配置。

5.2.2. 程序框架构建

在 Arduino IDE 中创建一个新的项目，定义所需的全局变量和常量，如 GPIO 引脚定义、激光测距仪的串口通信参数等。然后，编写初始化函数，用于设置硬件引脚和初始化串口通信。

5.2.3. 功能模块编写

根据系统设计，编写各个功能模块的代码。这包括：

- 按键扫描：编写函数来检测按键状态，并根据按键输入执行相应的操作。
- 激光测距仪通信：编写发送指令和接收数据的函数，处理与激光测距仪的通信。
- 数据处理：编写函数来解析激光测距仪返回的数据，并计算出距离。
- 用户界面更新：编写代码来更新 OLED 显示屏上的显示内容。
- 电源管理：编写代码来监控电池电量，并在电量低时发出警告。
- 错误处理：实现错误检测和处理机制，确保系统在遇到问题时能够给出提示。

5.2.4. 程序调试

在编写代码的过程中，需要不断地进行调试。这包括：

- 语法检查：确保代码没有语法错误，可以通过 IDE 的编译功能进行检查。
- 功能测试：对每个功能模块单独进行测试，确保它们按预期工作。
- 整体测试：将所有功能模块整合在一起，进行整体测试，确保系统功能完整。
- 性能优化：对程序进行优化，提高运行效率，减少资源占用。

5.2.5. 错误处理与日志记录

在程序中实现错误处理机制，当发生错误时，能够记录错误信息，并在 OLED 显示屏上显示。同时，可以通过串口输出错误日志，便于开发者分析问题。

5.2.6. 程序优化

在确保程序功能完整且无明显错误后，对程序进行优化。这可能包括代码重构、减少不必要的计算、优化内存使用等。

5.2.7. 结论

软件编程与调试是确保激光测距仪能够正确运行的关键步骤。通过构建清晰的程序框架、编写功能模块、进行细致的调试和优化，可以确保软件的稳定性和可靠性。此外，良好的错误处理和日志记录机制有助于快速定位和解决潜在问题。

第 6 章. 结论与展望

本课程设计实现了一款基于 PLS-K-100 激光测距模块和 ESP32 微控制器的高精度激光测距仪。通过详细的硬件设计、软件编程与调试，以及严格的测试流程确保产品的性能和可靠性。系统能够满足 100 米范围内的 ±2mm 高精度测量需求，具有良好的用户界面和低功耗设计，且能够在多种环境下稳定工作。

项目设计在达到预期设计目标的基础上，仍有进一步改进和扩展的空间。以下是一些可能的未来工作方向：

1. 无线通信功能：增加 Wi-Fi 或蓝牙通信功能，使测距仪能够与智能手机或平板电脑连接，实现远程控制和数据传输。
2. 数据处理与存储：集成数据存储功能，允许用户记录测量数据，并在需要时进行回溯分析。
3. 增强现实(AR)集成：结合 AR 技术，通过智能手机或平板电脑的摄像头，将测量结果实时叠加到现实环境中，提高用户体验。
4. 多语言支持：扩展用户界面，支持多种语言，以适应不同地区用户的需求。
5. 环境适应性增强：进一步优化设备的设计，提高其在极端温度和湿度条件下的稳定性。
6. 智能算法优化：利用机器学习算法对测量数据进行处理，提高测量精度和可靠性。
7. 商业化与市场推广：将产品推向市场，为建筑、装修、工业测量等领域提供更高效、便捷的测量工具。

通过这些改进和扩展，我们希望这款激光测距仪将能够更好地满足市场需求，为用户提供更加完善的测量解决方案。

第 7 章. 参考文献

[1] 激光测距模块 PLS-K-100 产品手册 [2] ESP32 开发文档 [3] Arduino 开发环境指南 [4] 电源管理芯片 TP5100 和 LP3220 的数据手册

7.1. 附录

- 硬件原理图
- PCB 布局图
- 软件源代码