浙江大学

本科实验报告

仪器自动校准和自检技术

课程名称：仪器系统设计

姓名：

学院：生物医学工程与仪器科学学院

专业：生物医学工程

学号：

指导老师：周泓

2025 年 6 月 7 日

浙江大学实验报告

专业：生物医学工程
姓名：
学号：
日期：2025 年 6 月 7 日
地点：

课程名称：仪器系统设计　指导老师：周泓　成绩：
实验名称：仪器自动校准和自检技术　实验类型：　同组学生姓名：

1. 为了确保校准参数的准确性，请分析可以采取的措施。

为了确保仪器系统中的校准参数的准确性，可以采取以下措施：

措施类别	具体内容
定期校准	仪器测量参数的准确性会受到使用寿命、温度、湿度、外部环境以及误用等多种因素的影响。因此，仪器必须定期进行校准，以保证其在预定精度下正常工作。
高精度标准源	**外部自动校准：**需要采用高精度的外部标准。在进行外部校准期间，校准常数要参照外部标准来调整。 **内部自动校准：**利用仪器内部的校准源将各功能、各量程按工作条件调整到最佳状态。实际上消除了环境因素对测量准确度的影响，补偿工作环境、内部校准温度和可能影响测量的其他因素的变化。
校准程序的自动化和智能化	**自动比较验证：**仪器系统可以自动对所得测试结果与已知标准进行比较，将测量的不确定性进行量化，验证测量仪器是否工作在规定的指标范围内。 **便捷操作：**传统仪器校准需要专业人员操作，并且校准后可能需要根据误差修正表进行修正，使用不便。自动校准则无需外部设备和连线，只需要按要求启动内部自动校准程序。
零点自动校准	仪器零点漂移是造成零点误差的主要原因之一。利用零点自动校准技术，在各个功能的不同量程上分别进行零校准，测量出零点漂移并存入校准存储器，在正常测量时进行修正。
校准参数的存储与保护	**非易失性存储：**校准后的参数一般都保存在 EEPROM 或 FLASH 等非易失性存储器中，以确保断电后数据不丢失。 **数据保护：**外部校准一旦完成，新的校准常数会被保存在测量仪器存储器的被保护区域内，且用户无法取得，这样就保护了由于偶然的调整对校准完整性的影响。制造商应提供相应的校准流程和校准软件。
浮点运算的优化	一般来说，校准参数实际都为浮点数。由于微处理器性能关系，浮点运算计算量大且占用资源多。尽可能地将浮点运算转化为定点运算，例如将参数乘上倍数（一般为 1000 倍）后进行保存，然后在计算中结果再除以倍数后就得到实际测量值。
环境因素的补偿	在环境差别较大的情况下工作时，内部自动校准可以消除环境因素对测量准确度的影响，补偿工作环境的变化、内部校准温度的变化和可能影响测量的其他因素的变化。

表 1: 确保校准参数准确性的措施

2. 编程比较一下采用浮点数和整数存储校准参数的资源占用的差别。

比较维度	浮点数存储	整数（定点数）存储
存储空间占用	通常一个 `float` 类型占用 4 个字节（32 位），一个 `double` 类型占用 8 个字节（64 位）。 例如，存储 10 个校准参数，使用 `float` 占用 $10 \times 4 = 40$ 字节；使用 `double` 占用 $10 \times 8 = 80$ 字节。	通常采用将校准参数乘以固定倍数（如 100、1000、10000 等），然后存储为整数。根据数值范围和所需精度，可选择 `short`（2 字节）、`int`（4 字节）等。 如所需精度不高，使用 `short` 仅需 $10 \times 2 = 20$ 字节。
处理器性能	浮点运算通常比整数运算慢得多，尤其是在没有浮点运算单元（FPU）的微处理器上。如果微控制器没有 FPU，浮点运算需要通过软件库来模拟，这会增加大量的指令周期。	整数运算速度快，指令周期少，即使在没有 FPU 的微处理器上也能高效执行。
程序存储器占用	浮点运算库（`libm`）通常体积较大，会显著增加程序代码的大小，从而占用更多的 Flash 存储器空间。	不需要庞大的浮点运算库，程序代码体积小，节省 Flash 空间。
RAM 占用	浮点运算可能需要更多的堆栈空间来存储中间结果和函数参数。	通常比浮点运算占用更少的堆栈和 RAM 空间。
功耗	浮点运算通常会消耗更多的电能，这对于电池供电的仪器是一个重要考虑因素。	整数运算功耗较低。
编程复杂性	相对简单，直接使用标准浮点类型和运算。	相对复杂，需要手动管理倍数，并在每次运算后进行相应的缩放。容易引入缩放错误或溢出。
精度	能够表示较大范围和较高精度的数值，但存在浮点数的精度问题（如舍入误差）。	精度取决于所选择的倍数和整数类型。可以实现任意所需的精度，但需要权衡整数类型的范围。

表 2: 浮点数与整数存储资源占用比较

编程示例（伪代码）：假设有一个校准参数 $P = 12.345$，需要存储和使用。

浮点数存储：

```
// 定义校准参数数组
float calibration_params[10];

// 存储
calibration_params[0] = 12.345f;

// 使用
```

```
float measured_value = 100.0f;
float calibrated_result = measured_value * calibration_params[0];
```

Listing 1: 浮点数存储示例

整数（定点数）存储：假设倍数为 1000。

```
// 定义校准参数数组
int calibration_params_int[10];
const int SCALE_FACTOR = 1000;

// 存储
calibration_params_int[0] = (int)(12.345f * SCALE_FACTOR); // 存储为 12345

// 使用
float measured_value = 100.0f;
// 在计算时转换为浮点数或使用定点数运算
float calibrated_result_float = measured_value *
    ((float)calibration_params_int[0] / SCALE_FACTOR);
```

Listing 2: 整数（定点数）存储示例

实际性能测试：为了验证理论分析，我们编写了一个 Python 程序来实际测试整型和浮点数运算的性能差异：

```
import time

def time_operation(operation_name, operation_func, count=100000000):
    """测试单个操作的性能"""
    print(f"正在测试: {operation_name}")
    start_time = time.perf_counter()
    operation_func(count)
    end_time = time.perf_counter()
    elapsed_time = end_time - start_time
    print(f"执行时间: {elapsed_time:.6f}秒")
    return elapsed_time

def int_multiply_int_op(count):
    """整数乘整数操作"""
    a, b = 10, 3
    for _ in range(count):
        result = a * b

def float_multiply_float_op(count):
    """浮点数乘浮点数操作"""
    a, b = 10.0, 3.0
    for _ in range(count):
        result = a * b

# 主测试函数
def main():
```

```
    count = 100000000 # 1亿次操作
    results = {}
    results['int*int'] = time_operation("整数乘法", int_multiply_int_op, count)
    results['float*float'] = time_operation("浮点数乘法", float_multiply_float_op, count)
```

Listing 3: Python 性能测试代码

实际测试结果：在相同的 PC 硬件平台上进行 1 亿次运算操作，测试结果如下：

运算类型	执行时间（秒）	每秒操作数	相对性能
整数 × 整数	2.057	48,621,514	1.00x（基准）
整数 × 浮点数	3.030	33,001,304	1.47x（慢 47%）
浮点数 × 浮点数	2.560	39,065,195	1.24x（慢 24%）
整数 ÷ 整数	2.782	35,941,340	1.35x（慢 35%）
整数 ÷ 浮点数	2.963	33,745,698	1.44x（慢 44%）
浮点数 ÷ 浮点数	2.492	40,124,925	1.21x（慢 21%）

表 3: Python 环境下整型与浮点数运算性能实测对比

测试结果分析：

- **整数运算最快**：纯整数乘法运算速度最快，作为性能基准
- **混合运算性能下降**：整数与浮点数的混合运算（如 int×float）性能下降最明显，比纯整数运算慢约 47%
- **浮点数运算适中**：纯浮点数运算性能介于纯整数和混合运算之间
- **除法比乘法慢**：无论是整数还是浮点数，除法运算都比相应的乘法运算慢
- **类型转换开销**：混合运算中的隐式类型转换是性能下降的主要原因

这个实际测试结果验证了理论分析：在资源受限的嵌入式环境中，使用整数（定点数）进行校准参数的存储和计算确实能够获得更好的性能表现。

3. 分析比较定时自检和实时自检的应用场景。

比较维度	定时自检（周期性自检）	实时自检（键控自检）
定义	仪器系统在运行过程中，不断地、周期性地插入自检操作。这种自检完全是自动进行的，并且是在测量工作的间歇期间完成的，不干扰正常测控任务。除非检查到故障，否则周期性自检不为操作人员所发觉。	有些仪器是在面板上设置一个自检按键，由操作者控制来启动自检程序。这种自检模式简单方便，可以在测控过程中寻找一个适当的时机进行自检，而又不干扰正常测控工作的进行。
执行时机	预设时间间隔（例如每隔几秒、几分钟或几小时）自动触发。在测量工作间歇期或系统空闲时执行。	由操作者手动触发，通常在系统处于空闲状态或在不需要测量数据的特定时间点。
自动化程度	无需人工干预，系统自主完成	需要操作者主动控制
可见性	正常运行时用户无法感知，只有检测到故障时才会报警	自检过程通常伴有显示或提示，让操作者了解自检状态和结果
持续性	能够持续监控仪器在长时间运行过程中的状态，及时发现渐发性故障	操作者可以根据需要随时进行自检
对正常工作的影响	在不影响主功能的前提下进行	如果在测量过程中启动，可能会暂时中断或影响测量任务

表 4: 定时自检与实时自检的比较

应用场景类别	定时自检	实时自检
运行特性	长期连续运行的仪器系统：如工业控制系统、医疗监护设备、环境监测站等，需要持续保证其可靠性	交互式或用户操作频繁的仪器：如实验仪器、诊断设备等，用户可以根据需要进行故障排查
可靠性要求	对运行时可靠性要求较高的系统：例如，在测量过程中需要确保数据准确性的仪器，如高精度传感器系统、数据采集系统	需要用户明确确认自检结果的场合：例如，在进行关键测量前，用户可以手动触发自检以确认仪器状态良好
人工干预程度	难以进行人工干预的远程或自动化系统：确保设备在无人值守情况下的正常运行	针对特定故障或在怀疑有故障时进行：用户可能在发现异常后，通过手动自检来验证故障是否存在
故障类型	主要针对渐发性故障：周期性检查可以发现由于长时间运行导致的性能逐渐下降或部件老化等问题	主要针对突发性故障或疑似故障的快速验证
检查对象	RAM、ROM、FLASH 的持续性完整性检查	面板显示装置自检、RAM 和 ROM 自检、输入/输出通道自检、总线自检以及键盘自检等

表 5: 定时自检与实时自检的应用场景

总结：定时自检侧重于**后台的、持续的、自动化的监控**，适用于需要高可靠性、长期运行且不希望频繁人工干预的系统，旨在发现渐发性故障。而实时自检（键控自检）则侧重于**用户控制的、按需的、交互式的诊断**，适用于用户需要主动排查故障或在特定时机验证仪器状态的系统。在实际应用中，两者往往结合使用，以提供更全面的故障检测和诊断能力。

4. 如何对于段码式液晶屏进行自检？

段码式液晶屏（Segment LCD）通常用于显示数字、简单的符号或预定义的图标，例如计算器、数字万用表、电表等。对其进行自检的主要目的是检查所有显示段是否能正常点亮或熄灭，以及显示驱动电路是否正常工作。

(1) 基本原理

对段码式液晶屏进行自检的方法通常是在仪器上电或执行自检程序时，通过软件控制显示所有段，然后依次熄灭所有段，让操作人员观察或通过光电传感器进行自动检测。

段码式液晶屏由多个独立的段组成，每个段由一个或多个引脚控制其点亮或熄灭。自检的核心思想是：通过编程控制液晶屏驱动芯片，使其在一段时间内点亮所有可用的段，然后熄灭所有段，或进行逐段点亮/熄灭的循环。

步骤	阶段名称	具体操作
1	初始化显示	在自检开始前，首先清空液晶屏显示，确保所有段都处于熄灭状态
2	全段点亮	通过向液晶屏驱动芯片发送特定的指令或设置相应的寄存器，使液晶屏上的所有可点亮的段（包括数字段、小数点、单位符号、图标等）全部点亮。保持全段点亮状态一段时间（例如 1-3 秒），以便操作人员观察。
3	全段熄灭（或图案切换）	接着，清空所有段，使液晶屏全部熄灭，或显示一个预设的测试图案（例如，交替点亮/熄灭）。保持熄灭状态或测试图案一段时间。
4	（可选）逐段点亮/熄灭	对于更详细的自检，可以编写程序逐个或分组点亮/熄灭每个段，以更精细地检查每个段的功能。例如，从 0 到 9 显示数字，并依次点亮小数点和单位符号。
5	结果判断	**人工观察：**最常见的方法是让操作人员目视观察液晶屏在全段点亮和熄灭（或显示测试图案）时是否有异常。 **自动检测：**对于自动化程度较高的仪器，可以集成光电传感器或图像识别模块，通过检测屏幕亮度或特定区域的像素状态来判断显示是否正常。
6	反馈提示	如果检测到异常，系统应发出声光报警或在其他显示介质上提示故障信息，例如”LCD ERROR” 或”DISPLAY FAULT”。如果自检通过，可以显示”PASS” 或正常进入工作界面。

表 6: 段码式液晶屏自检步骤

注意事项	说明
驱动 IC 兼容性	自检程序需要根据所使用的液晶屏驱动 IC 的寄存器定义和命令协议来编写
对比度与视角	在自检时，也要注意液晶屏的对比度和视角设置，以确保在不同观察条件下都能正确判断显示状态
时间控制	每个显示阶段的持续时间应足够长，以便观察
故障定位	如果发现某个段有问题，自检程序可以尝试提供更详细的信息，例如是哪个数字位或哪个特定符号的段出现问题，以便维修人员定位

表 7: 段码式液晶屏自检注意事项

(2) 基于 NXP PCF8553 的段码式液晶显示屏深度自检方法

有些时候显示屏坏了并不一定是显示屏的问题，可能是它背后的驱动芯片出问题了。虽然可能和题目原意关系不大，不过我还是出于个人兴趣基于 NXP PCF8553 数据手册，做了更加深入和系统化的段码式液晶显示屏自检方法研究。

1. 段码式液晶显示屏技术概述

段码式液晶显示屏（Segment LCD）因其固有的简洁性、低功耗特性和成本效益，在各类电子产品中得到了广泛应用。这类显示屏通过调控液晶材料的光学特性，以预定义的段（如数字、字母或特定符号）形式呈现信息，常见于家用电器、工业控制面板以及各类便携式电子设备中。

基本工作原理：段码式液晶显示屏的基本构造包括两层涂有氧化铟锡（ITO）的玻璃，其间精确夹持着一层扭曲向列（TN）液晶流体。当在段电极及其对应公共线之间施加交流（AC）电压时，该段内的液晶分子会随电场方向排列，改变光学性质，使该段呈现"ON"（暗）状态。

关键要求：对于液晶显示屏的长期可靠性而言，至关重要的一点是它们必须仅由交流电压驱动，确保任何段上的平均直流（DC）分量为零。长时间暴露于直流偏压会导致液晶流体发生不可逆的电化学降解，最终导致显示屏损坏。

2. 驱动器 IC 集成自检的优势

以 NXP PCF8553 为例，现代液晶驱动器 IC 提供了强大的自检能力：

IC 功能特性	自检优势
内部偏压生成	可通过软件调整偏压设置进行对比度诊断，无需外部电压调整
自动递增数据加载	简化复杂测试图案的生成，减少 CPU 开销和代码复杂性
多种多路复用模式支持	支持静态、2、3 或 4 背板多路复用，适应不同显示复杂度
可配置帧频率	提供 32Hz、64Hz、96Hz 和 128Hz 选项，用于闪烁检测和优化
I^2C/SPI 双接口	灵活的通信选择，适应不同开发环境和性能要求

表 8: PCF8553 驱动器 IC 的自检相关功能

3. 系统化自检程序

3.1 初始设置与安全检查在进行任何软件驱动的自检之前，必须执行严格的硬件级检查：

(1) **电源验证：**确保数字逻辑（VDD）和 LCD 驱动电压（VLCD）稳定，处于 1.8V 至 5.5V 范围内

(2) **电源时序：**绝对关键的是，LCD 驱动电压（VLCD）不能在数字电源电压（VDD）不存在时施加

(3) **未用段处理**：LCD 面板上任何未被驱动器主动驱动的段都应明确连接到其各自的背板（COM）引脚

(4) **通信接口设置**：验证 I²C 或 SPI 连接正确，包括必要的上拉电阻

3.2 基本连通性验证（万用表方法）对于未知 LCD 面板，可使用万用表进行引脚识别：

(1) 将万用表设置为二极管测试模式

(2) 将一个探头放在可疑的公共（COM）引脚上

(3) 系统地用另一个探头在其余引脚上移动，观察 LCD 段的亮起情况

(4) 记录引脚到段的映射关系，建立自定义引脚排列表

3.3 ”所有段开启”测试这是基础诊断测试，用于快速验证 LCD 上所有段的基本功能：

```
void pcf8553_all_segments_on_test() {
    // 1. 初始化PCF8553配置
    // Device_ctrl寄存器(01h): 设置帧频率
    I2C_write_register(0x01, 0x01); // 64Hz帧频率

    // Display_ctrl_1寄存器(02h): 设置偏压
    I2C_write_register(0x02, 0x00); // 1/3偏压

    // Display_ctrl_2寄存器(03h): 设置反转模式
    I2C_write_register(0x03, 0x00); // 线路反转

    // 2. 将0xFF写入所有DDRAM地址(04h至17h)
    for (int addr = 0x04; addr <= 0x17; addr++) {
        I2C_write_register(addr, 0xFF);
    }
}
```

Listing 4: 所有段开启测试伪代码

3.4 顺序段测试用于细致验证每个独立段的功能：

```
void pcf8553_sequential_segment_test() {
    // 逐个激活每个段
    for (int seg_index = 0; seg_index < 40; seg_index++) {
        // 清除所有段
        clear_all_segments();

        // 计算目标段的DDRAM地址和位位置
        int ddram_addr = 0x04 + seg_index / 8;
        int bit_pos = seg_index % 8;

        // 激活当前段
        I2C_write_register(ddram_addr, 1 << bit_pos);

        // 延迟观察
```

```
        delay_ms(200);
    }
}
```

Listing 5: 顺序段测试伪代码

3.5 对比度和闪烁性能测试

对比度调整：

- 通过修改 Display_ctrl_1 寄存器中的"B" 位调整偏压设置
- 支持 1/2 和 1/3 偏压配置
- 观察显示屏背景和激活段之间的清晰度差异

闪烁检测：

- 通过 Device_ctrl 寄存器的 FF[1:0] 位调整帧频率
- 人眼对 30Hz 以上闪烁敏感度降低，60Hz 以上通常无法检测闪烁
- 需在视觉质量和功耗之间找到平衡点

帧频率	功耗	闪烁情况	应用建议
32 Hz	最低	可能有可见闪烁	极低功耗应用，可接受轻微闪烁
64 Hz	适中	通常无闪烁	推荐的平衡设置
96 Hz	较高	无闪烁	对视觉质量要求高的应用
128 Hz	最高	确保无闪烁	高性能显示应用

表 9: PCF8553 帧频率设置对比

4. 故障诊断指南

故障现象	可能原因	诊断方法
部分段不亮	物理连接问题、DDRAM 映射错误	检查连接，验证映射表，用万用表测试
亮度暗淡不均	偏压设置不当、电源问题	调整偏压设置，检查电源电压稳定性
重影/串扰	未用段未连接 COM、过驱动	检查未用段连接，调整帧频率和电压
显示闪烁	帧频率过低	提高帧频率设置，平衡功耗需求
显示不稳定	通信问题、时序错误	检查 I²C/SPI 连接，验证时序参数

表 10: 常见故障现象与诊断方法

5. 最佳实践建议

(1) **分阶段测试：**从简单的" 所有段开启" 开始，逐步进行更详细的测试

(2) **参数优化：**利用 IC 的可配置参数，在显示质量和功耗间找到最佳平衡

(3) **环境适应：**在不同温度和电压条件下测试，确保稳定性

(4) **文档记录**：详细记录测试结果和配置参数，便于故障排除

(5) **定制化测试**：根据具体应用需求开发专门的测试模式

这种基于专用驱动器 IC 的系统化自检方法，不仅能够验证显示屏的基本功能，还能进行性能优化和故障诊断，确保段码式液晶显示屏在各种应用中的可靠运行。

5. 根据故障浴盆曲线原理分析仪器老化的必要性。

故障浴盆曲线（Bathtub Curve）是描述产品或系统在生命周期内故障率变化的经典模型。它将故障率曲线分为三个主要阶段：

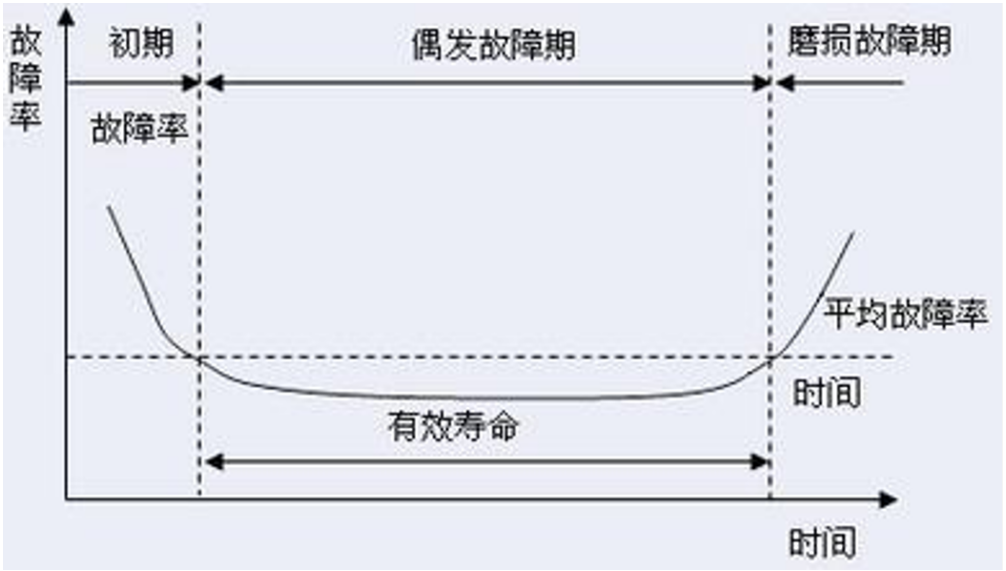

图 1: 故障浴盆曲线示意图

阶段名称	特征描述
早期故障期（高故障率，故障率迅速下降）	在产品投入使用初期，由于设计缺陷、制造缺陷、材料问题、运输损坏或安装不当等原因，故障率较高。随着时间的推移，这些”弱小”的或有缺陷的产品被筛选出来并修复或淘汰，故障率迅速下降。
偶发故障期（低故障率，故障率相对稳定）	在经过早期故障期后，产品进入稳定工作阶段。此时的故障主要由随机的、不可预测的因素引起，如环境突变、外部冲击、操作失误等。故障率保持在一个相对较低且稳定的水平。
耗损故障期（高故障率，故障率迅速上升）	随着产品使用时间的延长，磨损、疲劳、腐蚀、老化等因素开始显著影响产品的性能和可靠性。零部件逐渐老化失效，故障率开始迅速上升。

表 11: 故障浴盆曲线三个阶段

在仪器系统设计和制造过程中，进行”老化”测试（或称”烧机”、”高温老化”、”高低温循环”）其原理正是基于故障浴盆曲线的早期故障期。

老化作用	详细说明
筛选早期故障	**发现潜在缺陷：**在产品制造完成后，通常会对其进行一段时间的”老化”测试。这个过程会模拟仪器在实际工作环境（或更恶劣）下运行，通过高压、高温、高湿、长时间运行等应力条件，加速那些由于设计或制造缺陷引起的早期故障的显现。 **提高产品可靠性：**通过老化，可以提前发现并排除那些在早期故障期内就会失效的产品（”病态”产品）。这些产品在出厂前就被筛选出来，避免了在用户手中出现故障，从而显著提高了出厂产品的质量和可靠性。 **降低售后成本：**如果不进行老化测试，这些早期故障可能会在用户使用初期发生，导致大量的返修、更换和用户投诉，增加售后服务成本和品牌声誉损失。
将产品推入偶发故障期	老化测试的目标是让产品尽快地从早期故障期过渡到偶发故障期。这意味着当产品交付给用户时，它已经通过了最初的不稳定阶段，其故障率处于较低且相对稳定的水平。处于偶发故障期的产品更具可靠性，用户体验更好。
获取可靠性数据	老化测试过程中收集的故障数据有助于工程师分析故障模式，识别薄弱环节，并为后续的产品改进、优化设计提供数据和信息。此外，这些数据也可以用于预测产品的寿命和制定合理的维护计划。

表 12: 仪器老化的作用与必要性

总结：仪器老化（Burn-in 或 Aging）的必要性在于其能够主动、加速地暴露产品在生命周期早期可能存在的缺陷，从而在产品出厂前将其排除。这不仅提升了出厂产品的固有可靠性，减少了早期故障率，降低了售后成本，也使得产品在交付客户时已处于故障率较低的”偶发故障期”，从而为用户提供了更稳定、更可靠的使用体验。

6. 在汽车行驶过程中，轮胎压力异常是属于哪类故障？

根据故障分类标准，轮胎压力异常应从多个维度进行分析：

分类维度	故障类型	分析说明
按故障存在的程度分类	暂时性故障	如果是由于温度变化、载重变化等引起的临时压力异常，通过调整环境条件或充气可以恢复正常
	永久性故障	如果是由于轮胎刺破、气门嘴损坏等引起的持续漏气，必须更换或修复才能恢复正常
按故障发生、发展进程分类	**渐发性故障**	**主要特征：**轮胎压力异常通常是一个缓慢发展的过程，由轮胎缓慢漏气、橡胶老化、温度变化等因素引起。这类故障具有规律性，可通过轮胎压力监测系统（TPMS）进行早期监测和预防。
按故障严重程度分类	**非破坏性故障**	**主要特征：**轮胎压力异常在初期是渐发性的、局部性的，故障发生后暂时不会危及设备和人身安全。驾驶员有时间采取纠正措施。
按故障发生的原因分类	外因故障	扎钉子、路面异物刺破、人为操作不当（如过度充气或放气）等外部环境因素导致
	内因故障	气门嘴老化、轮胎橡胶材料缺陷、制造工艺问题等内部因素导致
按故障相关性分类	非相关故障	轮胎压力异常通常是轮胎本身的直接故障，而非由其他部件引起的间接故障

表 13: 轮胎压力异常的多维度故障分类

特殊情况说明：需要注意的是，如果轮胎发生**突然爆胎**，则属于：

- **突发性故障：**出现前无明显征兆，发生时间短暂
- **破坏性故障：**既是突发性又是永久性的，可能危及设备和人身安全

总结：轮胎压力异常最主要的分类特征是**渐发性故障**和**非破坏性故障**。它通常是一个可以监测、预防的逐渐发展过程，在故障初期不会立即造成严重危险，给驾驶员留有充分的反应和处理时间。

7. 结合具体的仪器故障，分析仪器故障的横向传播性和纵向传播性。

仪器故障的传播性是故障诊断中的重要概念，它描述了故障如何在系统内部蔓延。横向传播性指故障在同一层次内不同组件间的蔓延，而纵向传播性指故障在不同层次（元件、部件、子系统、系统）之间的传递。

(1) 横向传播性（Horizontal Propagation）：

指某一元件的故障引起同一层次内其他元件的功能失常。即在系统的某个功能层级或物理层级上，一个组件的故障影响到其相邻或相关联的同级组件。

故障实例	故障场景	横向传播分析
多通道数据采集系统中的电源模块故障	一个多通道数据采集卡，其板载电源模块为所有模拟输入通道提供供电。如果该电源模块中的某个电压稳压器发生故障，输出电压不稳定或超出正常范围。	所有依赖该电源模块供电的模拟输入通道（属于同一层级的功能模块）都会受到影响。例如，所有通道的测量结果可能出现漂移、噪声增大或测量值错误。这个故障并没有直接影响到更高层次的系统控制或更低层次的传感器本身，而是在”模拟输入通道”这一层级上横向传播。 **表现：**多个模拟量输入通道同时出现异常，但这些通道的输入信号源本身是正常的。
多路输出电源模块的短路保护失效	一个多路输出（如 +5V, +12V, -12V）的电源模块，其中一路（如 +5V）出现短路，但其短路保护机制失效。	短路可能导致整个电源模块的输出能力下降，甚至引起其他输出电压的跌落或波动。例如，+12V 和-12V 的输出也变得不稳定，影响到依赖这些电压的其他同级电路（如运算放大器、数字逻辑电路）。 **表现：**多个供电轨道上的器件同时出现异常或功能失常。

表 14: 横向传播性具体故障实例

(2) 纵向传播性（Vertical Propagation）：

指元件的故障相继引起部件 → 子系统 → 系统的故障。即故障从一个较低的系统层级向上蔓延到较高的层级，最终可能导致整个系统的失效。

故障实例	故障场景	纵向传播分析
温度传感器（元件）故障导致温度控制系统（系统）失控	一个精确的温度控制系统，由温度传感器（元件）、模拟信号调理电路（部件）、ADC 转换器（部件）、微控制器（子系统）和加热/冷却执行器（子系统）组成。如果温度传感器本身出现开路或短路故障。	**纵向传播过程：** 1. **元件故障：**温度传感器失效，无法准确输出温度信号。 2. **部件受影响：**模拟信号调理电路接收到错误的或无意义的信号。ADC 转换器对错误的信号进行数字化。 3. **子系统受影响：**微控制器根据错误的温度数据进行控制算法计算，导致输出的控制信号不正确。执行器接收到错误指令。 4. **系统故障：**整个温度控制系统无法维持设定温度，出现温度漂移或失控。 **表现：**最底层的元件故障导致了整个系统的功能性错误。
数据总线（部件）故障导致微处理器（子系统）无法加载程序（系统功能）	一个基于微处理器的仪器系统，其程序存储在外部 Flash 存储器中，通过数据总线与微处理器连接。如果数据总线上的某个数据线出现开路或短路故障。	**纵向传播过程：** 1. **部件故障：**数据总线上的数据线失效，导致数据传输错误。 2. **子系统受影响：**微处理器无法正确读取 Flash 中的程序指令和数据，导致程序运行异常、死机或崩溃。 3. **系统故障：**仪器系统无法正常启动或执行其功能。 **表现：**总线作为部件层的连接，其故障直接导致了上层微处理器子系统的工作异常，进而使得整个系统功能丧失。

表 15: 纵向传播性具体故障实例

传播性类型	特征总结
横向传播	故障在同一抽象或物理层级内，从一个组件蔓延到与其交互的其他同级组件
纵向传播	故障从较低的层级（元件）向上蔓延，逐步影响到更高层级的部件、子系统，最终导致整个系统的功能性失效

表 16: 故障传播性类型总结

8. 结合具体的仪器故障，分析比较基于信号处理和基于知识的故障诊断方法的异同点。

(1) 基于信号处理的故障诊断方法：

方法要素	内容描述
核心思想	基于信号分析理论，通过对系统运行时采集到的信号（如振动、电流、电压、温度、声学信号等）进行时域或频域分析，提取特征向量，然后利用这些特征向量与系统故障之间的关系来判断故障源的位置。
主要步骤	1. **信号采集：**采集与设备工作状态相关的各种物理量信号 2. **信号预处理：**对采集到的原始信号进行滤波、去噪、同步等处理 3. **特征提取：**利用各种信号处理技术（如傅里叶变换、小波变换、经验模态分解、频谱分析、倒谱分析等）从信号中提取能表征机器特征的信息，如均方根值、峰值、峭度、频率成分、能量分布等 4. **状态识别/故障判断：**将提取的特征向量与正常状态下的特征向量进行比较，或者根据预设的阈值、模式识别技术来判断是否存在故障，并识别故障类型和位置

表 17: 基于信号处理的故障诊断方法概述

故障实例	分析过程
旋转机械（如电机、风扇）的轴承故障诊断	**信号采集：**采集轴承附近的振动信号 **特征提取：**对振动信号进行频谱分析。在正常情况下，频谱图是平稳的；当轴承出现磨损、裂纹等故障时，会在特定频率（如轴承故障特征频率 BPFO、BPFI 等）出现谐波或边带，振动能量增大 **故障判断：**通过监测这些特征频率的幅值变化或趋势，判断轴承是否发生故障，以及故障的严重程度
电路板电源纹波异常检测	**信号采集：**采集电源输出端的电压信号 **特征提取：**对电压信号进行傅里叶变换，分析其纹波频率和幅值 **故障判断：**正常电源纹波应在允许范围内，且具有特定频率特征。如果出现异常频率成分或纹波幅值增大，则表明电源模块可能存在故障（如滤波电容失效）

表 18: 基于信号处理的故障诊断具体实例

评价维度	优点	缺点
适用性	适用于线性系统和非线性系统	仅仅是挖掘了系统信号中包含的信息，对于系统结构的信息并没有较好地利用
信息挖掘	能够挖掘系统信号中深层次的隐含信息	对传感器精度、采样频率和抗干扰能力要求较高
故障预警	适用于渐发性故障的早期预警和状态监测	依赖于对正常和故障信号特征的深入理解，特征提取和选择可能复杂

表 19: 基于信号处理方法的优缺点

(2) 基于知识的故障诊断方法：

方法要素	内容描述
核心思想	引入了被诊断系统的大量信息（如专家经验、维修手册、设计规范、历史故障数据等），通过推理、学习或模式匹配来判断故障。它不需要被诊断对象的数学模型。
主要类型	基于专家系统、基于神经网络、基于定性模型、基于模糊逻辑、基于模式识别、基于故障树等多种类型。机器学习算法是当前流行的方法。
主要步骤	1. **知识获取：**从专家、文档、历史数据中收集故障相关的知识，并以规则、案例、模型等形式表示 2. **特征提取（间接）：**从监测数据中提取与故障相关的符号化特征或高级信息（可能包括信号处理的结果） 3. **推理/学习：**根据获取的知识和提取的特征，进行逻辑推理、模式匹配或模型训练，以识别故障 4. **故障诊断：**输出故障类型、原因、部位和建议对策

表 20: 基于知识的故障诊断方法概述

故障实例	分析过程
医疗设备故障诊断专家系统	**知识库：**存储大量医疗设备的故障规则（如果 A 症状出现，并且 B 参数异常，则可能是 X 故障） **推理：**用户输入设备表现出的症状（如” 显示屏不亮”、” 测量数据跳动”），系统根据规则进行推理，给出可能的故障原因和排查建议
工业机器人故障诊断（基于机器学习）	**数据收集：**收集机器人多个传感器（电机电流、位置编码器、温度等）的历史运行数据，以及对应的故障标签（如” 关节 1 过载”、” 线缆断裂”） **模型训练：**使用机器学习算法（如支持向量机 SVM、神经网络 NN、决策树等）训练模型，使其能够从输入数据模式中识别出故障类型 **故障判断：**当新的运行数据输入时，训练好的模型能够预测是否存在故障以及具体的故障类型

表 21: 基于知识的故障诊断具体实例

评价维度	优点	缺点
模型依赖	不需要被诊断对象的精确数学模型	知识获取可能困难且耗时
系统复杂度	能够处理复杂、非线性的系统故障	模型的构建和训练可能需要大量数据和计算资源
信息融合	可以融合多种信息源（专家经验、历史数据）	解释性可能不如基于模型的方法直观（尤其对于复杂的黑箱模型）
适用场景	尤其适用于故障机理复杂、难以建立精确数学模型的系统	

表 22: 基于知识方法的优缺点

(3) 异同点比较：

比较维度	基于信号处理的方法	基于知识的方法
目的	都是为了实现故障的检测和诊断，提高仪器系统的可靠性、安全性和有效性	
流程	都包含信号采集、信号处理（或特征提取）、状态识别和诊断决策等步骤	
特征需求	都需要从原始数据中提取或识别出与故障相关的特征信息	
依赖模型	通常依赖于对信号的物理特性和数学模型的理解（虽然它仅挖掘信号信息，未利用系统结构信息）	不需要被诊断对象的精确数学模型，而是依赖于经验知识和数据
信息利用	主要侧重于挖掘信号本身的内在信息	引入了被诊断系统的大量外部信息（如专家经验、历史数据、故障规则等）
适用范围	更擅长分析周期性信号、动态特性变化等，适用于发现物理参数的变化	更擅长处理多源异构信息，适用于处理复杂、模糊、难以量化的故障
实现难度	可能需要专业的信号分析技术	可能需要构建庞大的知识库或训练复杂的机器学习模型

表 23: 两种故障诊断方法的异同点比较

总结：在实际应用中，这两种方法往往会结合使用，形成混合式故障诊断系统。例如，先利用信号处理方法提取底层特征，再将这些特征作为输入，通过基于知识的方法进行更高层次的故障推理和诊断。这种结合能够充分利用两者的优势，提高诊断的准确性和智能化水平。