同济大学

TONGJI UNIVERSITY

期末大作业报告

课题名称	虚拟发动机性能监控模块开发
副 标 题	大作业报告
学　　院	国豪书院
专　　业	计算机科学与技术
学生姓名	冉子易
学　　号	2352197
指导教师	陈宇飞
日　　期	2024 年 12 月 25 日

目 录

1 概览

1.1 功能描述

虚拟发动机性能监控模块旨在模拟并监控飞机发动机的各项性能指标，包括发动机转速、排气温度、燃油流速及燃油余量。模块需要根据不同的状态（如启动、运行、停车）动态生成数据，并实时更新这些数据。用户可以通过界面操作增加或减少推力，触发故障模式，并且在屏幕上显示告警信息。

1.2 设计思路

系统设计基于 C++ 编程语言，使用 EasyX 图形库进行界面绘制。核心功能包括：

- 数据模拟生成：根据设定规则模拟发动机转速、温度、燃油流速等数据的变化。
- 状态机管理：系统根据不同状态（启动、运行、停车）生成不同的数据，并根据用户输入进行推力调整。
- 异常检测与告警：系统可模拟 14 种异常情况，如传感器故障、燃油流速超标、温度过高等，并在界面上显示相应的告警信息。
- 图形界面：通过 EasyX 实现表盘显示、按钮控制及故障告警框的绘制。

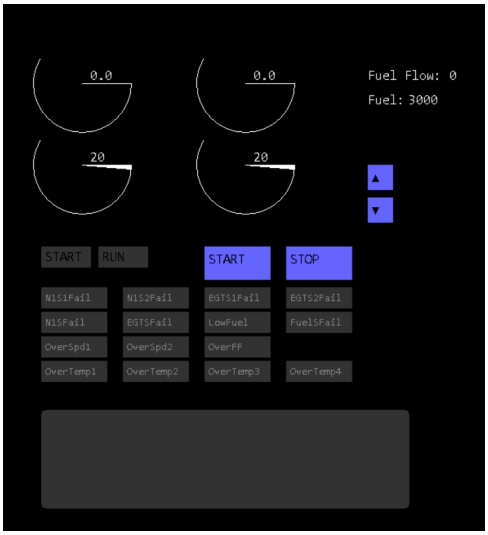


图 1.1 界面

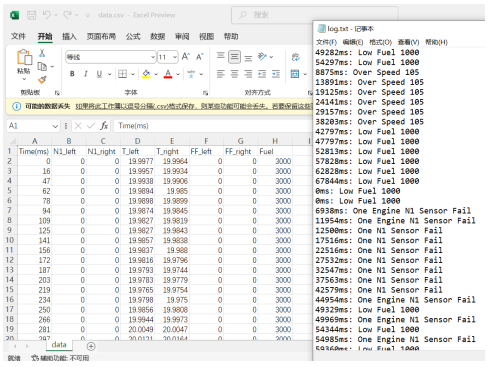


图 1.2 文件

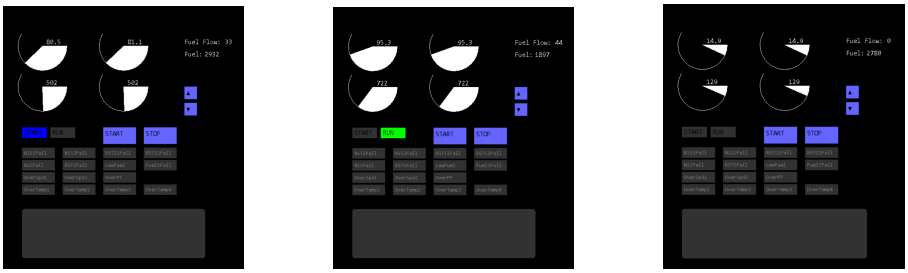


图 1.3 启动、稳态、停止

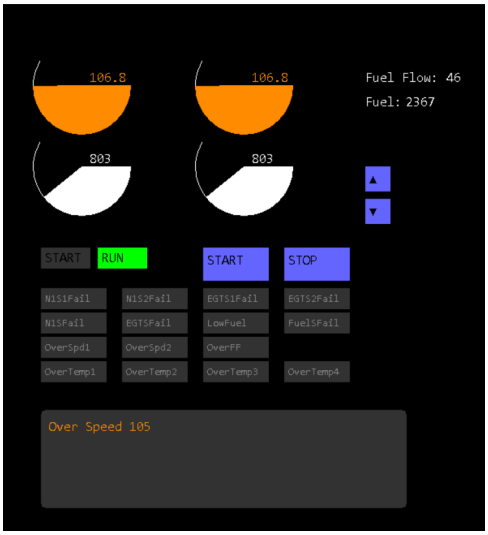


图 1.4 提升 + 超速 1

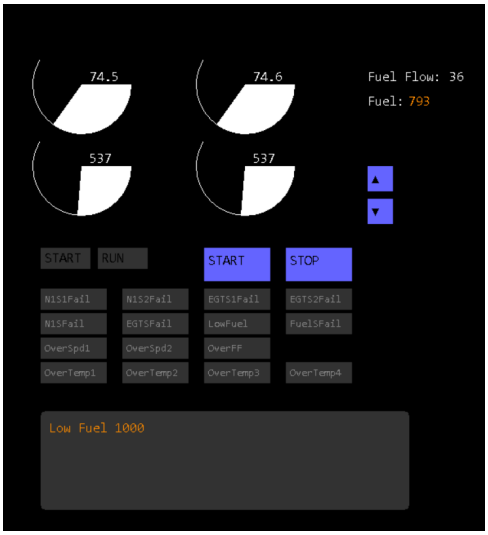
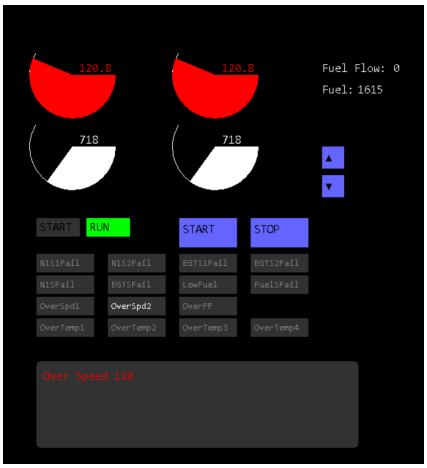
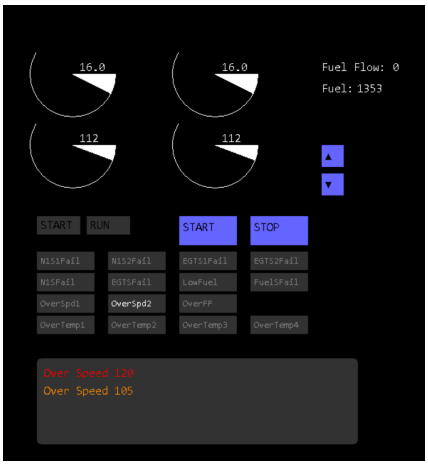


图 1.5 下降 + 低油



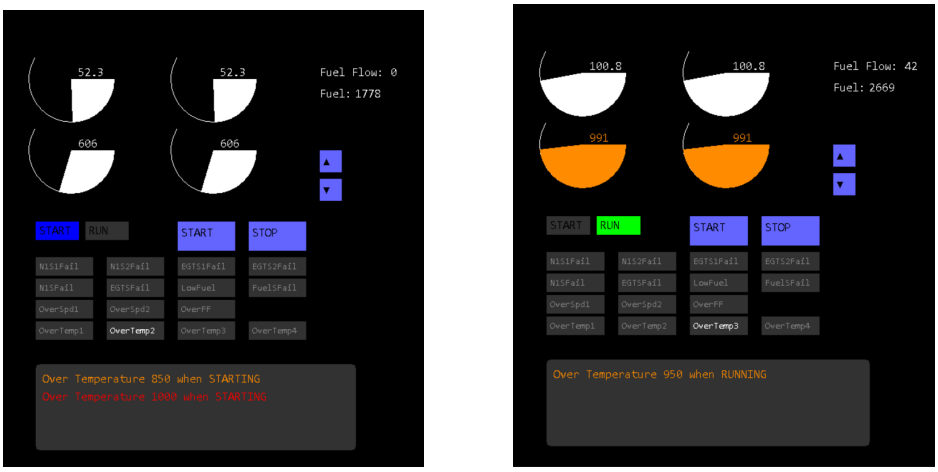
(a) 瞬间



(b) 停止

图 1.6 超速

装 订 线



(a) 启动 (b) 稳态

图 1.7 超温

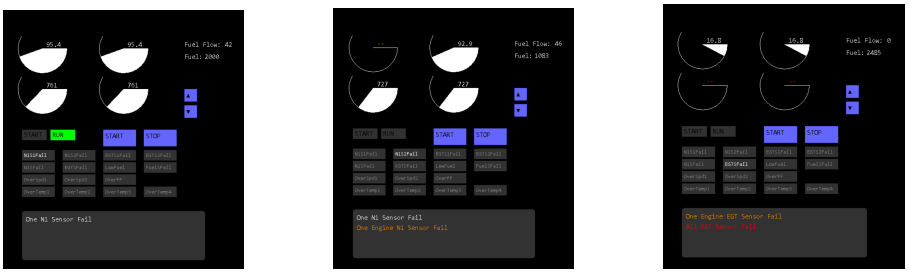


图 1.8 传感器

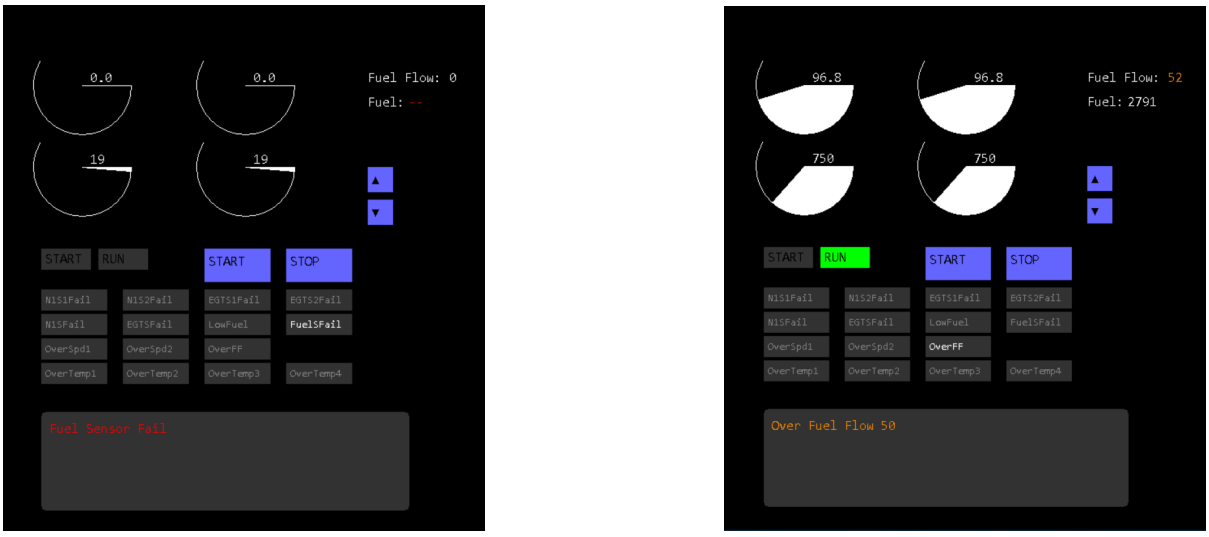


图 1.9 FSF 图 1.10 overFF

装 订 线

2 问题及解决方法

在开发虚拟发动机性能监控模块过程中，遇到了多个技术问题，并针对每个问题提出了解决方法。

2.1 问题一：告警信息重复显示

问题：在故障告警触发后，告警信息会重复显示在界面上，导致界面混乱。

解决方法：通过在 logFault 函数中增加对重复告警的检测，确保每个故障类型在 5 秒内不会被重复记录。同时，使用 g_alerts 向量来存储当前有效的告警信息，并在 drawTextArea 函数中只绘制有效的告警。

```
void logFault(FaultType ft) {
    if (ft == NO_FAULT) return;
    DWORD now = GetTickCount();

    // 遍历已记录的告警，检查是否已在5秒内记录
    for (auto& a : g_alerts) {
        if (a.type == ft) {
            if (now - a.last_trigger_time < 5000) return; // 防止重复记录
            a.last_trigger_time = now;
            break;
        }
    }

    // 新增告警
    AlertInfo ai;
    ai.type = ft;
    ai.text = faultTypeToString(ft);
    ai.color = getColorForFault(ft);
    ai.duration_ms = 5000;
    ai.last_trigger_time = now;
    g_alerts.push_back(ai);

    // 写入日志
    DWORD runningTime = now - g_startTime;
```

```
    g_logFile << runningTime << "ms: " << ai.text << std::endl;
    g_logFile.flush();
}
```

2.2 问题二：状态机切换问题

问题：在切换发动机状态时，数据更新和显示未能与状态变化同步，导致某些数据未能正确更新。

解决方法：通过状态机的设计，确保每次状态变化时，数据生成模块根据当前状态（如启动、运行或停车）正确生成数据。此外，在 ‘updateData‘ 中根据状态动态调整数据的变化速率，保证数据更新与状态变化同步。

```
void updateData() {
    DWORD now = GetTickCount(); // 获取当前时间
    double t = (now - g_startTime) / 1000.0; // 计算运行时间
    static double lastC = g_fuel.C;
    static DWORD lastUpdate = now;
    double dt = (now - lastUpdate) / 1000.0; // 计算时间增量
    lastUpdate = now;

    // 根据状态决定N,T,V变化
    if (g_state.state == ENGINE_OFF) {
        g_leftEngine.N1 = 0;
        g_rightEngine.N1 = 0;
        g_leftEngine.T = 20;
        g_rightEngine.T = 20;
        g_leftEngine.FF = 0;
        g_rightEngine.FF = 0;
    }
    else if (g_state.state == ENGINE_STARTING) {
        // 启动阶段: 转速、温度等的变化
        if (t < 2) {
            g_leftEngine.N1 += 10000.0 / 40000 * dt * 100; // 线性增加转速
            g_rightEngine.N1 = g_leftEngine.N1;
            g_leftEngine.FF += 5.0 * dt; // 燃油流速线性增加
            g_rightEngine.FF = g_leftEngine.FF;
```

```
            g_leftEngine.T = 20 + (g_leftEngine.N1 * 5); // 温度逐步增加
            g_rightEngine.T = g_leftEngine.T;
        }
    }
    // 更多状态逻辑
}
```

2.3 问题三：数据更新过于突然，缺乏平滑过渡

问题：发动机的转速、温度等数据变化过于突兀，缺乏平滑过渡，导致模拟不够真实。

解决方法：通过引入平滑因子（smoothingFactor）来控制数据的过渡，避免了数据的突变，使得系统更加平滑。

```
void updateData() {
    double smoothingFactor = 0.1; // 平滑过渡因子，控制数据变化速度

    // 平滑过渡到目标值
    g_leftEngine.N1 += (g_leftEngine.targetN1 - g_leftEngine.N1) * smoothingFactor;
    g_rightEngine.N1 = g_leftEngine.N1;

    g_leftEngine.FF += (g_leftEngine.targetFF - g_leftEngine.FF) * smoothingFactor;
    g_rightEngine.FF = g_leftEngine.FF;

    g_leftEngine.T += (g_leftEngine.targetT - g_leftEngine.T) * smoothingFactor;
    g_rightEngine.T = g_leftEngine.T;
}
```

2.4 问题四：故障信息显示不清晰

问题：故障告警信息的显示过于拥挤，难以清晰辨认。

解决方法：调整告警信息的显示方式，确保每条告警都有足够的空间显示。设置 ‘yOffset‘ 自动调整并限制显示区域，避免告警信息重叠。

```
void drawTextArea() {
    // 绘制背景框
    setfillcolor(RGB(50, 50, 50));
    solidroundrect(50, 500, 500, 620, 10, 10);
```

```
    settextstyle(20, 0, "Consolas");
    setbkmode(TRANSPARENT);
    DWORD now = GetTickCount();
    int yOffset = 510; // 初始位置

    // 显示告警信息
    for (const auto& alert : g_alerts) {
        if (now - alert.last_trigger_time > alert.duration_ms) continue; // 忽略过期的告警
        settextcolor(alert.color);
        outtextxy(60, yOffset, alert.text.c_str());
        yOffset += 25; // 调整位置
        if (yOffset > 590) break; // 超出范围时停止绘制
    }
}
```

3 心得体会

3.1 算法设计及编码过程

在本次项目中，最重要的部分是如何设计并实现数据生成模块，特别是如何根据不同的发动机状态来动态调整转速、温度、燃油流速等数据。通过使用数学模型（如对数增长模型）来模拟发动机的启动与停车过程，我对如何处理动态数据更新有了更深入的理解。此外，状态机的使用使得整个系统的逻辑更加清晰。

3.2 随机函数

为了增强系统的真实性，我使用了随机函数来对生成的数据进行小幅波动。例如，转速、温度和燃油流速在稳态下会有 ±3% 的波动。通过使用 ‘rand()‘ 函数生成随机数，并根据预设的波动范围调整数据，成功地模拟了发动机在实际工作中的不规则波动。

3.3 所思所感及吐槽

在实现过程中，最让我感到挑战的部分是故障模式的模拟与告警显示。尤其是如何设计一个清晰的界面，能有效地展示各种故障信息并避免重复显示。调试这部分逻辑时花了不少时间，因为告警信息的持续时间、是否重复记录、以及如何同步显示都需要精确控制。幸运的是，经过几轮调试，我最终解决了这些问题。

工作量是真的多。但我也是真的结课了。

=_=

4 部分源代码

```cpp
void logFault(FaultType ft) {
    if (ft == NO_FAULT) return;
    DWORD now = GetTickCount();

    // 遍历已记录的告警，检查是否已在 5 秒内记录
    for (auto& a : g_alerts) {
        if (a.type == ft) {
            if (now - a.last_trigger_time < 5000) return; // 防止重复记录
            a.last_trigger_time = now;
            break;
        }
    }

    // 新增告警
    AlertInfo ai;
    ai.type = ft;
    ai.text = faultTypeToString(ft);
    ai.color = getColorForFault(ft);
    ai.duration_ms = 5000;
    ai.last_trigger_time = now;
    g_alerts.push_back(ai);

    // 写入日志
    DWORD runningTime = now - g_startTime;
    g_logFile << runningTime << "ms: " << ai.text << std::endl;
    g_logFile.flush();
}
```

代码 4.1 日志记录故障信息并避免重复记录

```cpp
void drawTextArea() {
    // 绘制背景框
    setfillcolor(RGB(50, 50, 50));
    solidroundrect(50, 500, 500, 620, 10, 10);

    settextstyle(20, 0, "Consolas");
    setbkmode(TRANSPARENT);
    DWORD now = GetTickCount();
    int yOffset = 510; // 初始位置

    // 显示告警信息
    for (const auto& alert : g_alerts) {
        if (now - alert.last_trigger_time > alert.duration_ms) continue; // 忽略过期的告警
        settextcolor(alert.color);
        outtextxy(60, yOffset, alert.text.c_str());
        yOffset += 25; // 调整位置
        if (yOffset > 590) break; // 超出范围时停止绘制
    }
}
```

代码 4.2 绘制告警信息并限制显示范围

```cpp
void updateData() {
    DWORD now = GetTickCount(); // 获取当前时间
    double t = (now - g_startTime) / 1000.0; // 计算运行时间
    static double lastC = g_fuel.C;
    static DWORD lastUpdate = now;
    double dt = (now - lastUpdate) / 1000.0; // 计算时间增量
    lastUpdate = now;

    // 根据状态决定 N,T,V 变化
    if (g_state.state == ENGINE_OFF) {
        g_leftEngine.N1 = 0;
        g_rightEngine.N1 = 0;
        g_leftEngine.T = 20;
        g_rightEngine.T = 20;
        g_leftEngine.FF = 0;
        g_rightEngine.FF = 0;
    }
    else if (g_state.state == ENGINE_STARTING) {
        // 启动阶段：转速、温度等的变化
        if (t < 2) {
            g_leftEngine.N1 += 10000.0 / 40000 * dt * 100; // 线性增加转速
            g_rightEngine.N1 = g_leftEngine.N1;
            g_leftEngine.FF += 5.0 * dt; // 燃油流速线性增加
            g_rightEngine.FF = g_leftEngine.FF;
            g_leftEngine.T = 20 + (g_leftEngine.N1 * 5); // 温度逐步增加
            g_rightEngine.T = g_leftEngine.T;
        }
    }
    // 更多状态逻辑
}
```

代码 4.3 根据不同状态更新数据