高级语言程序设计（基础）课程实验报告

虚拟发动机性能监控模块

Virtual Engine Performance Monitoring Module

作者信息

姓名： 李天成

学号： 2451367

学院： 国豪书院

专业： 计算机科学与技术 (精英班)

完成日期：2026 年 1 月 4 日

目录

1. 题目简介

1.1 题目描述

发动机指示与机组告警系统（EICAS, Engine Indication and Crew Alerting System）是现代喷气式客机综合电子显示系统的重要组成部分。它主要用于向飞行员显示发动机的主要参数（如转速、温度、燃油流量等）以及飞机的各种系统告警信息。

本项目旨在基于 C++ 语言和 EasyX 图形库，开发一个双发喷气式飞机的 EICAS 仿真系统。该系统不仅需要实时模拟发动机在启动、运行、关机等不同阶段的物理参数变化，还需要具备完善的故障检测与告警功能，能够模拟传感器故障、超温、超速、燃油泄漏等多种异常工况，并根据故障等级（警告、警戒、提示）在界面上给出相应的视觉反馈。

1.2 系统功能要求

本仿真系统主要包含以下核心功能：

1. **物理仿真**：模拟双发动机（左发、右发）的 N1 转速、EGT（排气温度）和燃油系统的实时变化，包含启动过程的对数增长曲线和稳态运行时的随机物理波动。

2. **图形界面**：复刻真实的 EICAS 显示界面，包括模拟仪表盘、数字读数、状态指示灯以及 CAS（Crew Alerting System）消息显示区域。

3. **故障注入**：支持手动注入 14 种不同类型的故障，包括单/双传感器失效、燃油泵故障、发动机超温、超速等。

4. **智能告警**：根据传感器数据实时判断系统状态，生成不同优先级的告警消息（红色 Warning、琥珀色 Caution、白色 Advisory），并实现告警的抑制与清除逻辑。

1.3 告警逻辑定义

系统严格遵循航空工业标准的告警分级原则：

- **Level A (Warning)**：红色，表示需要立即采取行动的紧急情况（如发动机起火、严重超速）。

- **Level B (Caution)**：琥珀色，表示需要飞行员关注并采取行动的异常情况（如油量低、温度偏高）。

- **Level C (Advisory)**：白色，表示系统的状态改变或一般性信息。

2. 设计思路与整体架构

本程序采用了经典的 MVC（Model-View-Controller）变体架构，将物理仿真、逻辑控制与界面显示分离，确保了系统的高内聚低耦合。

图 1: 系统运行界面与架构示意

2.1 模块化架构设计

系统主要由以下三个核心模块组成：

- **EngineSimulator (Model)**：负责核心物理引擎的计算。它维护着系统的真实状态（SystemData），处理推力控制、燃油消耗以及故障对物理参数的影响。该模块不依赖于任何 UI 代码，仅负责数据的生成与更新。
- **AlertManager (Controller/Logic)**：负责业务逻辑与告警仲裁。它接收来自 Simulator 的传感器数据，根据预设的阈值规则进行故障检测，并管理告警消息的生命周期（触发、维持、消除）。该模块实现了迟滞（Hysteresis）逻辑以防止告警闪烁。
- **EngineUI (View)**：负责数据的可视化呈现。基于 EasyX 图形库，它将 Simulator 的数值和 AlertManager 的告警状态绘制到屏幕上，并处理用户的鼠标点击事件（如启动/停止、故障注入）。

2.2 数据流向与处理

系统的主循环遵循“输入-计算-渲染”的帧处理流程：

1. **输入处理**：UI 层捕获用户的鼠标操作（如调整推力、注入故障），并将指令传递给 Simulator。

2. **物理更新**：Simulator 根据当前状态（启动/运行）和时间步长（dt），计算下一帧的 N1、EGT 和燃油数据，并叠加随机波动噪声。

3. **告警检测**：AlertManager 分析最新的传感器数据，更新告警列表（Active Alerts）。

4. **界面渲染**：UI 层读取最新的系统数据和告警列表，刷新仪表盘指针、数字显示和 CAS 消息区域。

3. 实现细节

3.1 物理引擎与波动模拟

本系统的核心是一个基于状态机的物理仿真引擎（EngineSimulator）。为了真实还原航空发动机的运行特性，引擎将运行过程划分为三个独立阶段：启动（Starting）、稳态运行（Running）和关机（Stopping）。

3.1.1 分阶段模拟算法

- **启动阶段**：模拟涡扇发动机从静止加速到慢车转速的过程。系统采用对数增长函数模拟 N1 转速和 EGT（排气温度）的上升曲线，体现了物理惯性，避免了数值的突变。
- **稳态运行**：在此阶段，发动机参数由推力杆位置（Thrust Level）决定。系统根据当前的推力设定计算出目标 N1 和 EGT 值。
- **关机阶段**：模拟切断燃油后的惯性旋转停车过程，参数按指数衰减规律下降。

3.1.2 基于目标的随机波动模型

为了模拟真实机械仪表的“呼吸感”和传感器噪声，系统摒弃了简单的随机游走（Random Walk）算法，因为随机游走容易导致数值随时间发散。本系统采用了“基准值 + 噪声”的稳定模型：

$$V_{current} = V_{target} \times (1 + \text{Noise}(t)) \tag{1}$$

其中 $\text{Noise}(t)$ 是一个在 $[-3\%, +3\%]$ 区间内变化的随机因子。为了保证指针运动的平滑性，系统在每帧更新时并非直接跳转到含噪值，而是通过线性插值（Lerp）使当前值平滑趋向目标值，从而实现了既有波动又不会剧烈抖动的逼真效果。

Listing 1: 基于目标的波动与平滑更新逻辑

```
1 // EngineSimulator.cpp
2 void EngineSimulator::updateRunningPhase(double dt) {
```

```
    // 1. 计算含噪目标值 (Target + Noise)
    double noisyLeftN1 = addFluctuation(leftTargetN1, Constants
        ::FLUCTUATION_RANGE);
    double noisyLeftEGT = addFluctuation(leftTargetEGT,
        Constants::FLUCTUATION_RANGE);

    // 2. 执行平滑移动 (Lerp)
    // moveTowards 函数实现了 V_curr = V_curr + (V_target -
        V_curr) * rate * dt
    systemData_.leftEngine.n1Percentage = moveTowards(
        systemData_.leftEngine.n1Percentage,
        noisyLeftN1,
        n1Rate * dt
    );
}
```

3.2 告警管理与迟滞逻辑

告警管理模块（AlertManager）是系统的“大脑”，负责监控所有传感器数据并生成 CAS 消息。该模块最关键的技术创新在于引入了迟滞（Hysteresis）逻辑。

3.2.1 迟滞比较器的设计

在仿真系统中，由于引入了 3% 的随机波动，当参数值在告警阈值附近（如 $950°C$）波动时，会导致告警状态在“触发”和“消除”之间高频切换，造成界面闪烁。为此，我们实现了迟滞比较器：

- **触发阈值** ($T_{trigger}$)：例如，当 EGT 超过 $950°C$ 时，触发 Caution 告警。
- **清除阈值** (T_{clear})：只有当 EGT 回落到 $950 - \Delta$（如 $935°C$）以下时，才消除告警。

代码实现中，通过辅助函数 `isAlertActive` 检查当前状态，动态调整比较阈值：

Listing 2: 迟滞比较器实现代码

```
// AlertManager.cpp
void AlertManager::checkTempAbnormal(const EngineData& engine,
    AlertType type) {
    // 定义迟滞量 (Hysteresis)
    const double HYSTERESIS = 15.0;

```

```
    // 获取当前是否已处于告警状态
    bool isAlreadyActive = isAlertActive(type);

    // 迟滞逻辑判断
    if (!isAlreadyActive) {
        // 触发条件：超过上限
        if (engine.egt > Constants::EGT_MAX_LIMIT) {
            activateAlert(type);
        }
    } else {
        // 解除条件：低于（上限 - 迟滞量）
        // 只有当温度显著下降后才消除告警，防止临界值闪烁
        if (engine.egt < (Constants::EGT_MAX_LIMIT - HYSTERESIS)
            ) {
            deactivateAlert(type);
        }
    }
}
```

3.3 UI 渲染与交互系统

用户界面（EngineUI）基于 EasyX 图形库开发，采用了组件化设计思想。

3.3.1 双缓冲防闪烁技术

为了解决高频刷新（60FPS）带来的画面闪烁和撕裂问题，系统全面启用了双缓冲技术。在每一帧的渲染周期中：

1. 调用 `BeginBatchDraw()` 开启批量绘图模式，所有绘图指令写入内存显存；
2. 执行清屏、绘制表盘、绘制指针、绘制文字等操作；
3. 帧末调用 `FlushBatchDraw()` 将内存显存一次性拷贝至屏幕。

该机制确保了复杂仪表盘画面的稳定显示。

Listing 3: EasyX 双缓冲绘图流程

```
// EngineUI.cpp
void EngineUI::initialize() {
    initgraph(Constants::WINDOW_WIDTH, Constants::WINDOW_HEIGHT)
        ;
    // 开启双缓冲，防止画面闪烁
```

```
    BeginBatchDraw();
    // ... 资源加载 ...
}

void EngineUI::render(const SystemData& data, const std::vector<Alert>& alerts) {
    cleardevice(); // 清空缓冲区

    // 绘制各个图层
    drawBackground();
    drawGauges(data);
    drawAlerts(alerts);

    // 将缓冲区内容一次性输出到屏幕
    FlushBatchDraw();
}
```

3.3.2 矢量仪表绘制

系统中的 N1 和 EGT 表盘均采用参数化矢量绘制。通过极坐标转换公式，将归一化的物理数值映射为屏幕坐标：

$$\begin{cases} x = x_0 + r \cdot \cos(\theta) \\ y = y_0 - r \cdot \sin(\theta) \end{cases} \tag{2}$$

其中 θ 根据数值在 150° 至 360° 之间线性插值。此外，表盘的扇形区域颜色会根据当前的告警级别（Normal/Caution/Warning）动态改变，提供直观的视觉反馈。

3.4 故障注入机制

系统支持 14 种故障的实时注入，其实现原理是基于“数据源头污染”而非简单的界面欺骗。

- **传感器故障：**通过将 `SystemData` 中的 `sensorValid` 标志置为 `false`，导致 UI 层无法获取有效数据，从而显示”—” 或”FAIL” 标识。
- **物理故障：**例如“燃油流量过高”故障，并非直接修改显示的流量读数，而是在物理引擎计算层强制增加 `fuelFlow` 变量。这会引起物理模型的连锁反应——燃油流量增加导致燃烧加剧，进而导致 N1 转速上升和 EGT 温度升高，最终触发多重告警。这种深层模拟保证了故障现象的逻辑自洽性。

Listing 4: 故障注入对物理参数的影响

```cpp
// EngineSimulator.cpp
void EngineSimulator::injectFault(FaultType fault) {
    switch (fault) {
        case FaultType::FUEL_LEAK:
            // 燃油泄漏：直接减少物理模型中的燃油量
            // 这会导致后续计算中剩余油量加速下降
            systemData_.fuelQuantity -= 5.0 * dt;
            break;

        case FaultType::EGT_OVERHEAT:
            // 模拟过热：强制提升目标温度
            // 物理引擎会自动追随这个新的错误目标值
            leftTargetEGT = 1080.0; // 目标值远高于红色阈值(1000)，确保触发Danger
            break;

        case FaultType::SENSOR_FAIL:
            // 传感器失效：标记数据无效
            // UI层检测到此标记后会显示琥珀色叉号
            systemData_.leftEngine.sensorValid = false;
            break;
    }
}
```

3.5 安全保护与强制停车逻辑

为了模拟真实飞机的安全保护机制，系统实现了分级响应策略：

- **Warning (黄色)**：仅显示告警，不干预发动机运行。
- **Danger (红色)**：触发紧急停车程序。

特别地，针对“手动故障注入”场景，为了方便教学演示，系统实现了智能延迟停车逻辑：当用户手动注入严重故障（如超温）时，系统会允许物理参数继续爬升并超过 Danger 阈值，直到参数真正接近故障设定的极端目标值（如 $1080°C$）时才触发强制停车。这确保了用户能清晰观察到红色告警状态和仪表盘的极端读数。

Listing 5: 智能强制停车逻辑

```cpp
// main.cpp
if (highestLevel == AlertLevel::DANGER) {
```

```
    bool shouldStop = false;
    if (g_simulator->isFaultActive()) {
        // 手动故障模式：等待物理参数完全达到目标值才停车
        // 容差控制：N1 ±2%, EGT ±15度
        if (g_simulator->isFaultTargetReached()) {
            shouldStop = true;
        }
    } else {
        // 自然故障模式：一旦触发Danger立即停车
        shouldStop = true;
    }

    if (shouldStop && !g_simulator->isStopping()) {
        g_logger->recordEvent(timestamp, "CRITICAL: EMERGENCY
            SHUTDOWN");
        g_simulator->stopEngine();
    }
}
```

4. 遇到的问题及解决方法

4.1 仿真数值的随机游走问题

问题描述：在早期的物理引擎实现中，为了模拟仪表的抖动效果，采用了在上一帧数值基础上累加随机增量的算法（$V_{t+1} = V_t + \Delta$）。这种简单的随机游走（Random Walk）模型导致了严重的数值漂移现象，长时间运行后，发动机参数会逐渐偏离物理目标值，甚至出现负转速等不合逻辑的情况。

解决方法：重构了波动算法，改为“基于目标的噪声模型”。系统首先根据推力杆位置计算出确定的物理目标值（Target），然后在此基础上叠加 ±3% 的高斯白噪声，最后通过线性插值（Lerp）计算当前帧的显示值。这种方法确保了数值始终收敛于物理真实值附近，既保留了视觉上的真实波动感，又保证了系统的数值稳定性。

4.2 临界值附近的告警闪烁

问题描述：当引入随机波动后，如果发动机参数恰好处于告警阈值（如 EGT 950°C）附近，叠加的噪声会导致数值在阈值上下高频跳动。这导致 CAS 告警消息和仪表盘颜色在“正常”和“警告”状态之间快速切换，产生严重的频闪现象，极大地干扰了用户的视觉体验。

解决方法：在告警逻辑中引入了迟滞（Hysteresis）机制，即施密特触发器原

理。为每个关键参数设置触发阈值（T_{high}）和清除阈值（T_{low}）。例如，EGT 告警在超过 950°C 时触发，但必须回落到 935°C 以下才会消除。这 15°C 的安全缓冲区（Buffer Zone）有效过滤了仿真噪声引起的误触发，确保了告警状态的稳定性。

4.3 UI 布局与交互设计的迭代

问题描述：在初始设计中，UI 布局存在多处不合理：

- 按钮布局混乱，缺乏逻辑分组，导致操作不便。
- 状态指示灯（RUN Light）逻辑简单，仅判断 $N1 > 95\%$ 即点亮，导致在临界点附近频繁闪烁。
- 故障测试按钮的标签与实际触发逻辑不匹配（如标签显示 > 850 但实际触发 > 900），造成测试困惑。
- 故障注入功能最初仅通过少数几个按钮循环切换，无法精确控制特定故障，不利于测试。

解决方法：

- **网格化布局与功能拆分：**将故障注入功能从简单的循环切换重构为全功能的控制面板。设计了 4×4 的按钮网格，将 14 种具体的故障类型（如单发传感器失效、双发失效、燃油泄漏、各级超温等）独立拆分为单独的按钮，并按功能模块（传感器、燃油、转速、温度）进行逻辑分组，极大提升了测试效率。
- **指示灯迟滞：**为 RUN 指示灯添加迟滞逻辑（点亮阈值 95%，熄灭阈值 90%），确保状态显示的稳定性。
- **逻辑校准：**全面审查并修正了所有测试按钮的触发逻辑，确保按钮标签（如"ST > 950"）与后台注入的故障参数（980°C）及告警系统的阈值严格对应。

4.4 指针运动的平滑处理

问题描述：在引入随机波动后，如果直接将计算出的含噪数值赋给仪表盘指针，指针会出现剧烈的机械跳动，这不符合真实物理仪表的阻尼特性，视觉效果非常生硬。

解决方法：在 UI 渲染层引入了线性插值（Linear Interpolation, Lerp）算法。系统不再直接显示当前的物理计算值，而是维护一个“显示值”变量，每帧以一定的速率（如 0.1 的插值系数）向“物理目标值”逼近。

$$V_{display} = V_{display} + (V_{target} - V_{display}) \times \alpha \tag{3}$$

这种处理模拟了机械指针的惯性和阻尼，使得指针在响应数值变化时既灵敏又平滑，完美复刻了真实航空仪表的动态质感。

4.5 告警阈值的物理合理性修正

问题描述：初期设定的 EGT 告警阈值（850°C）过低。在正常巡航推力（N1 95

解决方法：参考真实航空发动机数据，调整了告警逻辑。将 Warning 阈值提升至 950°C，Danger 阈值提升至 1000°C。这既保留了对异常工况的敏感度，又为正常的大推力运行留出了合理的物理裕度（Margin），消除了误报警现象。

5. 心得体会

5.1 跨学科知识的融合与应用

本次实验不仅仅是一次编程练习，更是一次跨学科的学习之旅。为了实现逼真的 EICAS 仿真，我深入查阅了航空发动机的相关资料，了解了 N1 转速、EGT 排气温度、燃油流量等参数之间的物理关联，以及它们在启动、慢车、巡航等不同阶段的变化规律。这让我深刻体会到，优秀的软件工程师不仅需要掌握编程技术，更需要具备快速学习和理解业务领域知识（Domain Knowledge）的能力。只有真正理解了业务背后的物理模型，才能写出逻辑严密、仿真度高的代码。

5.2 工程规范与鲁棒性设计

在开发告警系统的过程中，我切身感受到了工业级软件对“鲁棒性”的严苛要求。简单的阈值判断在理想环境下或许可行，但在充满噪声和波动的真实（或仿真）环境中，必须引入迟滞逻辑、信号滤波等机制来保证系统的稳定性。此外，告警分级（Warning/Caution/Advisory）的设计也让我理解了人机交互中信息分层的重要性——在紧急情况下，必须通过最直观的颜色和方式将最关键的信息传递给操作者。这些工程规范和设计原则，对于我未来从事任何领域的软件开发都是宝贵的经验。

5.3 面向对象设计的实践感悟

通过将系统拆分为 Simulator（物理模型）、AlertManager（逻辑控制）和 EngineUI（视图显示）三个独立模块，我再次验证了高内聚、低耦合架构的优势。这种设计使得我在后期频繁调整告警阈值或修改 UI 布局时，完全不需要改动物理引擎的核心代码，极大地降低了维护成本和出错概率。这次实践让我对面向对象设计原则（SOLID）有了更直观和深刻的理解。