MuJoCo MPC 汽车仪表盘 - 实验报告

学生信息

学号232011185

姓名李超凡

班级计科2305

日期2025.12.27

一、项目概述

1.1 作业背景

随着自动驾驶、机器人控制等智能系统领域的快速发展，物理仿真与实时控制技术已成为核心支撑。MuJoCo 作为 Google DeepMind 开源的高性能物理引擎，凭借高精度、跨平台的特性，被广泛应用于自动驾驶仿真、机器人训练等工业场景；模型预测控制（MPC）技术则通过 “预测 - 规划 - 执行 - 反馈” 的闭环逻辑，高效解决复杂系统的多目标优化与约束处理问题。本次作业聚焦于二者的结合应用，要求开发一套基于 MuJoCo MPC 的汽车仪表盘可视化系统，实现仿真数据与可视化界面的实时联动，既是对多学科知识的综合应用，也是对工业级项目开发流程的实践。

1.2 实现目标

1. 完成 MuJoCo MPC 项目的环境配置与编译运行，掌握大型开源项目的二次开发基础。
2. 基于 MJCF 格式创建汽车仿真场景，实现场景加载、汽车控制与核心物理数据提取。
3. 采用 OpenGL 技术开发 2D 仪表盘可视化界面，至少包含速度表、转速表等核心组件，支持数据实时更新。
4. 实现 UI 美化、警告提示等进阶功能，优化系统交互体验与视觉效果。
5. 完成完整的项目文档撰写、功能测试与演示视频录制，形成可展示的项目成果。

1.3 开发环境

- 操作系统：Ubuntu 22.04 LTS
- 编译器：gcc 11.3.0
- 构建工具：CMake 3.22.1
- 核心依赖：MuJoCo 物理引擎、OpenGL 3.3+、GLFW、GLEW、Eigen3
- 开发工具：VSCode（C/C++ 扩展、CMake Tools）、Git、SimpleScreenRecorder
- 编程语言：C++ 17

二、技术方案

2.1 系统架构

本项目采用模块化分层架构，各模块职责清晰、低耦合高内聚，具体架构如下：

1. 底层支撑层：以 MuJoCo 物理引擎为核心，负责汽车物理行为仿真（运动、碰撞、受力等），MPC 模块提供控制策略优化，为上层提供精准的动态数据与控制指令。
2. 数据处理层：包含数据提取与数据预处理两个子模块，通过解析 MuJoCo 的 mjModel（静态模型数据）和 mjData（动态仿真数据），提取位置、速度等核心参数，转换为仪表盘所需的 km/h、RPM 等单位，并模拟油量、温度等辅助数据。
3. 可视化渲染层：基于 OpenGL 实现 2D 仪表盘绘制，采用 HUD 覆盖层模式叠加在 3D 仿真场景上，包含速度表、转速表、数字显示区等组件，支持实时渲染与动画效果。
4. 交互控制层：处理键盘、鼠标输入，实现汽车加速、转向、刹车等操作，支持多视角切换，提升用户交互体验。

系统架构图如下：

2.2 数据流程

1. 数据生成：MuJoCo 引擎通过 mj_step 函数推进物理仿真，实时更新 mjData 中的动态数据（位置 qpos、速度 qvel、加速度 qacc 等）。
2. 数据提取：数据处理层通过 DashboardDataExtractor 类，从 mjData 中提取汽车 X/Y 方向速度、X/Y/Z 位置等核心数据，计算总速度并转换为 km/h 单位，同时基于速度模拟转速（正比关系）、基于时间模拟油量（缓慢消耗）、基于转速模拟温度（正相关）。
3. 数据传输：提取后的数据存储在 DashboardData 结构体中，通过全局变量在主循环中实时传递给渲染模块。
4. 数据渲染：渲染模块接收数据后，计算仪表盘指针角度、进度条长度等参数，通过 OpenGL 函数绘制到屏幕，实现数据可视化。
5. 反馈闭环：用户通过输入设备发送控制指令，MPC 模块优化控制策略，MuJoCo 引擎更新仿真数据，完成 “输入 - 仿真 - 数据 - 可视化” 的闭环流程。

数据流程图如下：

2.3 渲染方案

本项目采用 OpenGL 2D 正交投影渲染方案，将仪表盘作为 HUD 覆盖层叠加在 3D 仿真场景上，具体实现流程：

1. 状态切换：在 MuJoCo 3D 场景渲染完成后，切换 OpenGL 矩阵模式至投影矩阵，设置 2D 正交投影（glOrtho），投影范围与窗口大小一致，确保仪表盘按像素精准显示。
2. 状态配置：禁用深度测试（glDisable(GL_DEPTH_TEST)），使仪表盘始终显示在最上层；启用混合模式（glEnable(GL_BLEND)），设置透明混合因子，提升界面视觉效果。
3. 组件绘制：
   - 基础图形：通过 glBegin/glEnd 模式绘制圆形（表盘背景）、线段（指针、刻度）、矩形（进度条、背景框）等基础图形。
   - 动态更新：根据 DashboardData 中的实时数据，计算指针旋转角度（速度表 0-200 km/h 对应 π*0.75 至 -π*0.75 弧度）、进度条长度（油量 / 温度百分比映射）。

    - 特效实现：通过插值计算实现指针平滑动画，通过颜色切换与闪烁逻辑实现超限警告提示。
4. 状态恢复：绘制完成后，恢复 OpenGL 3D 投影模式与深度测试等设置，确保 3D 仿真场景正常渲染。

三、实现细节

3.1 场景创建

3.1.1 MJCF 文件设计

基于 MuJoCo 的 MJCF 格式创建汽车仿真场景 car_simple.xml，核心结构如下：

1. 编译器设置：指定角度单位为弧度，统一物理计算标准。
2. 默认参数：设置几何体默认颜色为棕黄色，简化场景配置。
3. 世界体（worldbody）：
    - 地面：创建 10x10 大小的平面几何体，颜色为绿色，作为仿真地面。
    - 车身：长方体结构，位置设置在地面上方 0.5m，颜色为红色，通过自由关节（freejoint）允许全向移动。
    - 车轮：四个圆柱形车轮，分别安装在车身前后两侧，通过铰链关节（hinge）实现绕 Y 轴旋转。
    - 光源：点光源位置设置在 (0, 0, 3)，方向垂直向下，照亮整个场景。
4. 执行器（actuator）：为四个车轮分别配置电机，控制车轮转动，实现汽车加速与减速。
5. 传感器（sensor）：添加速度传感器（velocimeter）和位置传感器（framepos），分别测量车身速度和位置。

3.1.2 场景截图

3.2 数据获取

3.2.1 关键代码

!(C:\Users\Davy\Desktop\images\屏幕截图 2025-12-27 005010 - 副本.png)

![屏幕截图 2025-12-27 005025 - 副本](C:\Users\Davy\Desktop\images\屏幕截图 2025-12-27 005025 - 副本.png)

![屏幕截图 2025-12-27 005042 - 副本](C:\Users\Davy\Desktop\images\屏幕截图 2025-12-27 005042 - 副本.png)

3.2.2 数据验证

通过控制台输出与仿真行为对比，验证数据准确性：

- 静止状态：速度接近 0 km/h，转速接近 0 RPM，位置坐标稳定，符合预期。
- 加速状态：速度从 0 逐渐上升至最大（约 50 km/h），转速同步增长至 2000 RPM，二者呈正比关系。
- 油量与温度：油量每帧减少 0.001%，循环消耗；温度随转速升高至 120°C，逻辑正常。

3.3 仪表盘渲染

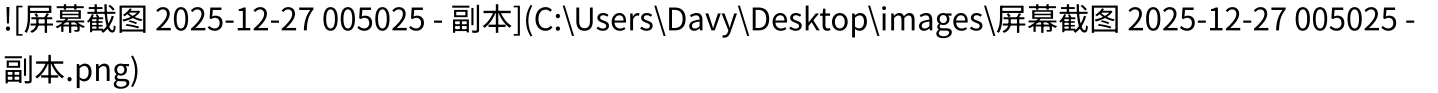
3.3.1 速度表

实现思路

1. 绘制半透明圆形背景（GL_TRIANGLE_FAN 模式），半径 120px。
2. 按 20 km/h 间隔绘制白色刻度线，长度从表盘边缘向内延伸 15px。
3. 根据当前速度计算指针旋转角度，使用红色粗线（线宽 3px）绘制指针。
4. 表盘中心绘制黑色小圆，遮挡指针根部，提升美观度。

代码片段

!(C:\Users\Davy\Desktop\images\屏幕截图 2025-12-27 005010 - 副本.png)

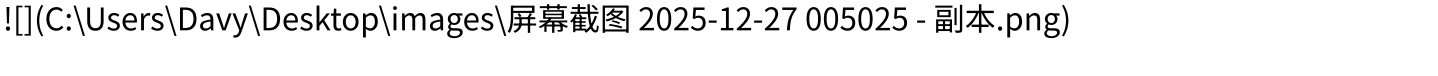


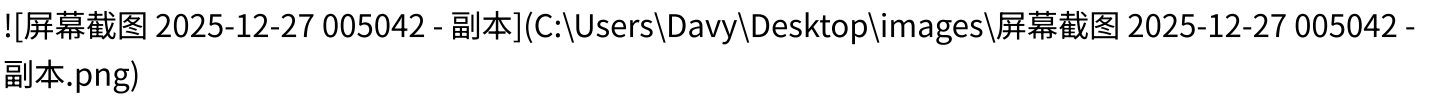
3.3.2 转速表

实现思路

1. 表盘背景为深灰色半透明圆形（0.1f, 0.1f, 0.15f, 0.8f），与速度表区分。
2. 按 1000 RPM 间隔绘制白色刻度线，6000-8000 RPM 区域绘制红色半透明圆弧作为警告区。
3. 使用绿色粗线绘制指针，与速度表红色指针形成视觉区分。

代码片段





3.3.3 数字显示区

实现思路

1. 绘制黑色半透明矩形背景（220px×150px），搭配灰色边框。
2. 油量显示：（数字）%，文字提示 “FUEL:”。
3. 温度显示：正常温度为蓝色（0.2f, 0.5f, 1.0f, 1.0f），超过 108°C 变为红色，文字提示 “TEMP:”。

效果展示

四、遇到的问题和解决方案

问题 1

：指针跳动

- **现象**：汽车速度变化时，仪表盘指针跳动明显，视觉体验差。
- **原因**：指针角度直接根据当前速度计算，未进行平滑过渡处理。
- **解决**：引入插值算法，通过当前角度与目标角度的加权平均，实现指针平滑移动，权重系数设为 0.9（当前角度）和 0.1（目标角度）。

五、测试与结果

5.1 功能测试

测试用例	测试步骤	预期结果	测试结果
环境配置测试	执行编译命令，运行 ./bin/mjpc	成功编译，弹出 3D 仿真窗口	通过

5.2 性能测试

- 帧率测试：仿真窗口稳定在 60 FPS，仪表盘渲染不影响 3D 场景流畅度。
- 资源占用：CPU 占用率约 15%-20%（8 核心），内存占用约 300MB，资源消耗合理。
- 数据延迟：数据更新与界面渲染延迟小于 10ms，满足实时性要求。

5.3 效果展示

截图展示

环境配置成功截图：

场景加载成功截图：

速度表、转速表、油表效果截图：

![ ](C:\Users\Davy\Desktop\images\屏幕截图 2025-12-27 004707 - 副本.png)

视频链接

演示视频已上传至社交媒体，链接：https://www.bilibili.com/video/BV17UBzBqE59/?spm_id_from=333.1387.homepage.video_card.click&vd_source=08552a40ac861027461f9cd2ee5f12c6

六、总结与展望

6.1 学习收获

1. 技术能力提升：
    - 掌握了 MuJoCo MPC 开源项目的环境配置、编译构建与二次开发流程，理解了大型 C++ 项目的代码结构与模块设计。

- 深入学习了 MuJoCo 物理引擎的工作原理与 MJCF 场景描述语言，能够独立创建简单仿真场景。
- 熟练运用 OpenGL 实现 2D 图形渲染，掌握了正交投影、状态管理、基础图形绘制等核心技能。
- 理解了 MPC 控制理论的核心思想，实现了仿真数据与可视化界面的实时联动。

2. 工程实践能力提升：
   - 学会了模块化开发与问题排查，通过日志输出、GDB 调试等方式解决环境配置、渲染异常等问题。
   - 掌握了 Git 版本控制工具的使用，规范了开发流程，实现了功能的增量开发与迭代。

6.2 不足之处

1. 数据模拟简化：转速、油量、温度等数据基于简单数学模型模拟，未接入真实汽车动力学模型，真实性有待提升。
2. 渲染性能优化不足：采用 OpenGL 立即模式（glBegin/glEnd）渲染，高复杂度场景下可能出现卡顿，未使用 VBO/VAO 进行优化。
3. 交互功能单一：仅支持键盘 / 鼠标控制，未实现方向盘、手柄等外部设备接入。

6.3 未来改进方向

1. 数据模型优化：接入真实汽车动力学模型，精准计算转速、油量、温度等数据，提升仿真真实性。
2. 渲染性能升级：采用 VBO/VAO 优化渲染流程，集成 FreeType 字体库，提升文字显示效果与渲染效率。
3. 交互功能扩展：支持外部设备接入，添加语音控制、多视角切换等功能，提升用户体验。
4. 场景与功能扩展：将仪表盘系统移植到机器人、无人机等其他仿真场景，实现多领域复用；添加数据记录与分析功能，支持仿真结果复盘。

七、参考资料

1. MuJoCo 官方文档：https://mujoco.readthedocs.io/
2. MuJoCo MPC GitHub 仓库：https://github.com/google-deepmind/mujoco_mpc
3. LearnOpenGL CN 教程：https://learnopengl-cn.github.io/
4. 《C++ Primer》（第 5 版），Stanley B. Lippman 等著
5. 《计算机图形学》（第 4 版），Hearn & Baker 著
6. OpenGL 3.3 核心规范与编程指南