

# MuJoCo MPC 汽车仪表盘 -作业报告

## 一、项目概述

### 1.1 作业背景

物理仿真技术的崛起 随着人工智能、自动驾驶、机器人技术的快速发展，物理仿真引擎在现代科技领域中扮演着越来越重要的角色。传统的实体验证方法成本高昂、风险大，而基于物理仿真的虚拟测试成为行业标准解决方案。MuJoCo 作为 DeepMind 开源的先进物理引擎，已经成为机器人学习、自动驾驶仿真等领域的行业标准工具。 模型预测控制的工业应用 模型预测控制(MPC) 作为现代控制理论的重要分支，因其能够处理多变量、有约束的复杂系统，在工业过程控制、自动驾驶、机器人运动规划等领域得到广泛应用。将 MPC 与物理仿真结合，可以实现在虚拟环境中测试和优化控制算法，大幅降低开发成本和风险。

### 1.2 实现目标

#### 1.2.1 核心基础目标

物理仿真环境构建 成功配置 MuJoCo MPC 开发环境，确保在 Ubuntu/Linux 系统下正常编译运行 理解 MJCF 文件格式，创建包含完整车辆模型的仿真场景 实现车辆基本运动控制，包括加速、转向、制动等基础动力学模拟 实时数据流管道建立 从 MuJoCo 仿真引擎中实时提取车辆状态数据 实现数据格式转换和单位标准化（如 m/s 到 km/h 的转换） 建立稳定的数据更新机制，确保仪表盘显示与仿真状态同步 基础仪表盘可视化 基于 OpenGL 实现速度表和转速表的基础渲染 完成 2D UI 层在 3D 场景上的正确叠加显示 确保数据展示的实时性和准确性，误差控制在合理范围内

#### 1.2.2 功能完善目标

多维度数据监控 实现速度、转速、油量、温度等关键参数的并行监控 开发数据预警机制，对异常状态进行可视化提示 添加数据历史记录功能，支持简单的趋势分析 用户交互体验优化 实现仪表盘布局的合理规划和视觉美化 添加必要的交互元素，如视角切换、数据显示控制等 优化渲染性能，确保系统在主流硬件上流畅运行 系统稳定性保障 完成边界情况测试，确保系统在各种工况下的稳定性 实现错误处理机制，对异常输入和环境变化具有容错能力 优化资源管理，避免内存泄漏和性能下降问题 能力培养目标

#### 1.2.3 技术能力提升

大型项目开发能力 掌握阅读和理解工业级 C++代码的能力 学会在现有开源项目基础上进行功能扩展和二次开发 培养从需求分析到系统实现的完整项目开发流程经验 跨技术栈整

合能力 物理引擎（MuJoCo）与图形渲染（OpenGL）的技术整合 实时系统设计与性能优化  
经验的积累 多线程/实时数据处理的实践能力培养 工程实践能力 版本控制（Git）、持续集成等现代开发工具的使用 调试技巧和性能分析工具的掌握 技术文档编写和项目演示的能力训练

## 1.3 开发环境

- 操作系统：Ubuntu 22.04
- 编译器：gcc 11.3.0
- CMake：3.22.1

## 二、技术方案

### 2.1 系统架构

#### 系统架构说明

系统整体采用“仿真 + 控制 + 可视化”三层结构：

- **仿真层**：MuJoCo 负责物理建模与动力学计算
- **控制层**：mjpc 提供模型预测控制接口
- **可视化层**：
  - 3D 场景中的仪表盘渲染（基于 mjvGeom）
  - 控制台终端文本仪表显示

#### 模块划分

- MJCF 场景描述模块
- 车辆状态获取模块
- 仪表盘渲染模块
- 终端状态输出模块
- 能耗（油量）计算模块

---

### 2.2 数据流程

#### 数据流程说明

系统中的数据流向如下：

1. MuJoCo 根据模型与控制输入计算物理状态
2. 仿真结果存储在 mjData 结构体中
3. 从 mjData 中读取车辆位置、速度、加速度等信息
4. 将数据分别传递给：
  - 3D 仪表盘渲染模块
  - 控制台终端显示模块
  - 油耗计算模块
5. 实时更新并显示结果

## 数据结构设计

- data->qpos: 车辆位置（广义坐标）
  - data->qvel: 车辆速度
  - data->qacc: 车辆加速度
  - data->ctrl: 控制输入（油门、转向）
  - 静态变量用于累计油耗与状态统计
- 

## 2.3 渲染方案

### 渲染流程

- 使用 MuJoCo 提供的 mjvScene 结构
- 在 ModifyScene() 回调函数中动态添加几何体
- 仪表盘由多个几何体组合而成（圆环、刻度、指针、标签）

### OpenGL 使用说明

本项目未直接调用底层 OpenGL 绘制接口，而是通过 MuJoCo 提供的高级可视化接口 mjvGeom 间接完成渲染。该方式可以保证渲染结果与仿真坐标系一致，简化开发流程。

## 三、实现细节

### 3.1 场景创建

- MJCF 文件设计

#### MJCF 文件设计

- 使用 MJCF 描述汽车模型与环境
- 车辆由多个几何体组合而成（车身、轮子等）
- 定义必要的传感器用于速度信息获取



## 3.2 数据获取

### 关键代码说明

通过 MuJoCo 的 `mjData` 结构体实时获取车辆状态：

- 位置： `data->qpos`
- 速度： `data->qvel`
- 加速度： `data->qacc`
- 车体速度： 通过传感器 `car_velocity`

### 数据验证方式

- 在终端中实时打印数值
- 观察车辆运动与数值变化是否一致
- 通过静态输出验证数据连续性与合理性

---

## 3.3 仪表盘渲染

### 3.3.1 速度表

#### 实现思路

- 根据车辆线速度计算速度比例
- 将速度映射到  $180^\circ$  的仪表盘角度范围
- 使用指针几何体表示当前速度

#### 代码片段

```
double* car_pos = data->xpos + 3 * car_body_id;
```

```
// 仪表盘位置（汽车正前方，立起来）
```

```
float dashboard_pos[3] = {  
    static_cast<float>(car_pos[0]),  
    static_cast<float>(car_pos[1] ), // 汽车前方0.5米  
    static_cast<float>(car_pos[2] + 0.3f) // 地面上方0.3米  
};
```

```
// 最大速度参考值（km/h），根据要求是0-10
```

```
const float max_speed_kmh = 10.0f;
```

```
// 速度百分比（0-1）
```

```
float speed_ratio = static_cast<float>(speed_kmh) / max_speed_kmh;  
h;
```

```
if (speed_ratio > 1.0f) speed_ratio = 1.0f;
```

```
// 仪表盘旋转矩阵（绕X轴旋转90度，再顺时针旋转90度）
```

```
double angle_x = 90.0 * 3.14159 / 180.0; // 绕X轴旋转90度（立起
```

效果展示



### 3.3.2 转速表

#### 实现思路

- 使用车辆速度近似模拟发动机转速
- 将转速映射为固定长度的终端字符条（30 格）
- 使用 # 表示当前转速水平段，- 表示其他水平段

#### 代码片段

```
const int BAR_LEN = 30;

const double max_speed_ref = 5.0;  // 参考最大速度
double rpm_ratio = speed_ms / max_speed_ref;
if (rpm_ratio > 1.0) rpm_ratio = 1.0;
if (rpm_ratio < 0.0) rpm_ratio = 0.0;

int filled = static_cast<int>(rpm_ratio * BAR_LEN);

char rpm_bar[BAR_LEN + 1];
for (int i = 0; i < BAR_LEN; i++) {
    rpm_bar[i] = (i < filled) ? '#' : ' ';
}
rpm_bar[BAR_LEN] = '\0';
```

效



果

展

示

---

### 3.4 进阶功能

- 原地刷新终端输出，避免刷屏

- 油耗模型与剩余油量百分比显示
  - 终端文本仪表与 3D 仪表盘数据同步
- 

## 四、遇到的问题 and 解决方案

### 问题 1: MuJoCo MPC 编译失败

#### 编译时出现错误

CMake Error at CMakeLists.txt:105 (find\_package): Could not find a package configuration file provided by "mujoco" with any of the following names: mujocoConfig.cmake mujoco-config.cmake

#### 原因分析:

MuJoCo 依赖库未正确下载或编译 CMake 缓存文件过期或损坏 网络问题导致依赖下载不完整

#### 解决方案:

```
cd ~/mujoco_projects/mujoco_mpc rm -rf build mkdir build && cd build export
GIT_REPO_BASE="https://gitee.com/mirrors" cmake .. -DCMAKE_BUILD_TYPE=Release
-DCMAKE_VERBOSE_MAKEFILE=ON make -j4 2>&1 | tee build.log
```

### 问题 2: OpenGL 上下文初始化失败

Failed to create OpenGL context GLFW error 65543: GLX: Failed to create context

#### 原因分析:

显卡驱动不兼容或过时 系统缺少必要的 OpenGL 库 WSL 环境下未正确配置图形支持

#### 解决方案:

```
sudo ubuntu-drivers autoinstall sudo apt install nvidia-driver-535 # NVIDIA 显
卡 sudo apt install mesa-utils libgl1-mesa-dri libglu1-mesa glxinfo | grep
"OpenGL version"
```

## 五、测试与结果

### 5.1 功能测试

#### 测试用例

- 车辆直线行驶
- 车辆加速与减速
- 控制输入为零时状态变化

#### 测试结果

- 仪表盘显示与车辆运动状态一致
  - 终端数据显示稳定、连续
  - 油量百分比随时间合理变化
- 

### 5.2 性能测试

- 仿真运行流畅，无明显卡顿
  - 仪表盘渲染未对仿真性能造成明显影响
  - 终端输出对帧率影响较小
- 

### 5.3 效果展示

- 场景运行截图
- 仪表盘效果截图
- 演示视频链接

#### MJCF 场景展示

[点击观看演示视频](#)

---

## 六、总结与展望

### 6.1 学习收获

- 熟悉了 MuJoCo 的数据结构与仿真流程
  - 掌握了仿真数据到可视化结果的完整实现方法
  - 提高了对 C++ 工程结构与调试能力的理解
- 

## 6.2 不足之处

- 油耗模型为简化模型，未考虑真实发动机特性
  - 仪表盘样式仍较为基础
  - 缺少更复杂的交互功能
- 

## 6.3 未来改进方向

- 引入更真实的车辆动力学与能耗模型
  - 优化仪表盘视觉效果与动画表现
  - 增加数据记录与分析功能
  - 扩展为多车辆或多场景仿真系统
-