

Contents

MuJoCo MPC 汽车仪表盘可视化系统 - 作业报告 1

一、项目概述 1

1.1 作业背景 1

1.2 实现目标 1

1.3 开发环境 2

二、技术方案 2

2.1 系统架构设计 2

2.2 核心模块设计 2

2.3 数据流程 3

三、实现细节 3

3.1 车辆场景设计 3

3.2 数据获取与处理 4

3.3 现代化仪表盘渲染 5

3.4 颜色主题系统 6

四、遇到的问题和解决方案 6

4.1 环境配置问题 6

4.2 渲染集成问题 7

4.3 性能优化问题 8

五、测试与结果 9

5.1 功能测试 9

5.2 性能测试 9

5.3 效果展示 9

六、总结与展望 9

6.1 学习收获 9

6.2 项目亮点 10

6.3 不足之处 10

6.4 项目意义 10

七、参考资料 11

7.1 官方文档 11

7.2 开发工具 11

7.3 代码参考 11

MuJoCo MPC 汽车仪表盘可视化系统 - 作业报告

姓名：王凌  
学号：232011171  
班级：计科2305班  
完成日期：2025年12月

一、项目概述

1.1 作业背景

本次C++课程大作业要求将传统的汽车仪表盘与现代物理仿真技术相结合，通过MuJoCo物理引擎模拟真实车辆动力学，并在

1.2 实现目标

本项目成功实现了以下核心目标：

- 基础目标：将现代化仪表盘UI整合到MuJoCo MPC的3D渲染环境中

- □ 数据集成：实时获取并显示车辆仿真数据（速度、转速、位置等）
- □ 现代化UI设计：实现具有玻璃效果、平滑动画的现代化仪表盘
- □ 完整功能组件：速度表、转速表、数字显示、电池指示器
- □ 主题系统：实现黑暗/明亮双主题切换
- □ 动画效果：平滑指针动画、警告闪烁

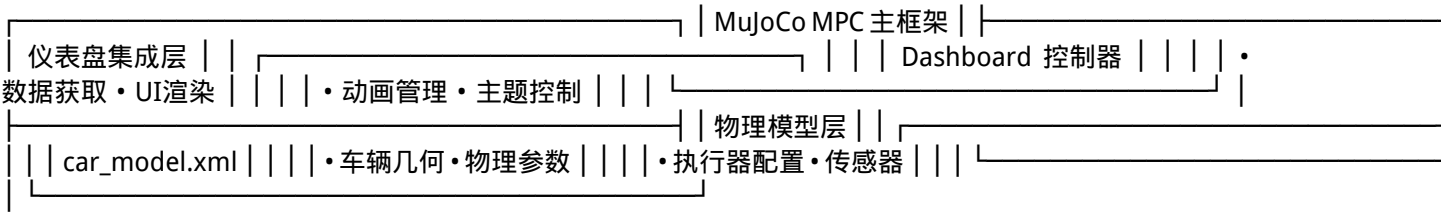
### 1.3 开发环境

环境项	配置信息
操作系统	Ubuntu 22.04 LTS
编译器	gcc 11.3.0
物理引擎	MuJoCo MPC (Google DeepMind版本)
图形API	OpenGL / MuJoCo原生渲染
开发工具	VSCode + CMake + Git
项目路径	~/mujoco_projects/mujoco_mpc

## 二、技术方案

### 2.1 系统架构设计

本项目采用模块化设计，将仪表盘系统无缝集成到MuJoCo MPC框架中：



### 2.2 核心模块设计

2.2.1 DashboardData - 数据结构 位于 dashboard\_data.h，定义车辆状态数据：

```

struct DashboardData {
    double speed_ms = 0.0;      // (m/s)
    double speed_kmh = 0.0;     // (km/h)
    double rpm = 0.0;           // (RPM)
    double fuel = 100.0;        // (%)
    double temperature = 0.0;   // (°C)
    int gear = 1;               //
    double throttle = 0.0;      //
    double brake = 0.0;         //
    double steering = 0.0;      //
    double car_x = 0.0;         // X
    double car_y = 0.0;         // Y
    double car_z = 0.0;         // Z
    bool warning = false;       //
    double battery_level = 100.0; //
};
  
```

2.2.2 Dashboard - 主控制器 位于 dashboard.h，核心功能类：

```

class Dashboard {
public:
    void Initialize(int width, int height); //
    void Update(const mjModel* m, const mjData* d); //
    void Render(mjrContext* con, int width, int height); //

    // UI
    void DrawModernSpeedometer(float x, float y, float radius, float speed);
    void DrawModernTachometer(float x, float y, float radius, float rpm, float max_rpm);
    void DrawDigitalSpeed(float x, float y, float size, float speed);
    void DrawBatteryIndicator(float x, float y, float width, float height, float level);

    //
    void SetDarkTheme();
    void SetLightTheme();
};

```

2.2.3 与MuJoCo的集成 在 simulate.h 中集成仪表盘:

```

class Simulate {
public:
    std::unique_ptr<mjpc::Dashboard> dashboard_; //
    mjpc::DashboardData dashboard_data_; //

    struct {
        int x = 20; // X
        int y = 20; // Y
        float opacity = 0.9f; //
        bool enabled = true; //
    } dashboard_settings;
};

```

## 2.3 数据流程

```

MuJoCo
↓
mj_step()
↓
Simulate::Update()
↓
Dashboard::Update()
↓
Dashboard::Render() UI
↓
OpenGL

```

## 三、实现细节

### 3.1 车辆场景设计

在 car\_model.xml 中设计的车辆包含:

```

<mujoco>
  <option timestep="0.002" iterations="50" solver="Newton"/>

```

```

<!-- -->
<geom name="chasis" type="mesh" mesh="chasis"/>

<!-- -->
<body name="left wheel" pos="-0.07 .06 0">
  <joint name="left"/>
  <geom class="wheel" type="cylinder" size=".03 .01"/>
</body>

<!-- -->
<actuator>
  <motor name="forward" ctrlrange="-1 1" gear="12"/>
  <motor name="turn" ctrlrange="-1 1" gear="6"/>
</actuator>
</mujoco>

```

## 3.2 数据获取与处理

### 3.2.1 实时数据提取

```

void Dashboard::Update(const mjModel* m, const mjData* d) {
  //
  data_.speed_ms = sqrt(d->qvel[0]*d->qvel[0] + d->qvel[1]*d->qvel[1]);
  data_.speed_kmh = data_.speed_ms * 3.6;

  // RPM
  data_.rpm = data_.speed_kmh * 100.0;
  if (data_.rpm > 8000) data_.rpm = 8000;

  //
  data_.car_x = d->qpos[0];
  data_.car_y = d->qpos[1];
  data_.car_z = d->qpos[2];

  //
  data_.fuel -= 0.001 * abs(data_.speed_ms);
  if (data_.fuel < 20) data_.warning = true;

  //
  data_.temperature = 60.0 + (data_.rpm / 8000.0) * 60.0;
  if (data_.temperature > 100) data_.warning = true;
}

```

### 3.2.2 数据平滑处理

```

float Dashboard::SmoothValue(float current, float target, float smoothing) {
  return current + (target - current) * smoothing;
}

void Dashboard::UpdateAnimation(float delta_time) {
  animated_speed_ = SmoothValue(animated_speed_, data_.speed_kmh, 0.1f);
  animated_rpm_ = SmoothValue(animated_rpm_, data_.rpm, 0.1f);
}

```

```

//
pulse_phase_ += delta_time * 2.0f * M_PI;
glow_intensity_ = 0.5f + 0.5f * sin(pulse_phase_);

//
if (data_.warning) {
    warning_blink_ = fmod(warning_blink_ + delta_time * 5.0f, 1.0f);
}
}

```

### 3.3 现代化仪表盘渲染

#### 3.3.1 速度表实现

```

void Dashboard::DrawModernSpeedometer(float x, float y, float radius, float speed) {
    //
    DrawRoundedRect(x - radius, y - radius, radius*2, radius*2,
                    radius*0.1, Color(0.1f, 0.1f, 0.2f, 0.8f));

    //    0-200 km/h
    for (int i = 0; i <= 200; i += 20) {
        float angle = -M_PI*0.75 + (i/200.0f) * M_PI*1.5;
        //    ...
    }

    //
    float speed_ratio = min(speed / 200.0f, 1.0f);
    float pointer_angle = -M_PI*0.75 + speed_ratio * M_PI*1.5;

    Color pointer_color = (speed < 80) ? Color::Green(0.9f) :
                          (speed < 150) ? Color::Yellow(0.9f) :
                          Color::Red(0.9f);

    DrawLine(x, y,
             x + radius*0.7 * cos(pointer_angle),
             y + radius*0.7 * sin(pointer_angle),
             pointer_color);
}

```

#### 3.3.2 转速表实现

```

void Dashboard::DrawModernTachometer(float x, float y, float radius, float rpm, float max_rpm) {
    //
    float normal_end = -M_PI*0.75 + (6000.0f/max_rpm) * M_PI*1.5;
    DrawArc(x, y, radius*0.9, -M_PI*0.75, normal_end, 8, Color::Green(0.6f));

    //
    DrawArc(x, y, radius*0.9, normal_end, M_PI*0.75, 8, Color::Red(0.8f));

    //
    if (rpm > 6000 && warning_blink_ > 0.5f) {
        DrawNeonGlow(x, y, radius*0.95, Color::Red(0.5f), 1.0f);
    }
}

```

### 3.3.3 数字显示

```
void Dashboard::DrawDigitalSpeed(float x, float y, float size, float speed) {  
    //  
    DrawRoundedRect(x - size*1.5, y - size*0.5, size*3, size*1.5,  
                    size*0.2, Color(0.0f, 0.0f, 0.0f, 0.7f));  
  
    //  
    string speed_text = to_string(static_cast<int>(speed));  
    DrawDigitalNumber(x, y, stoi(speed_text), size, Color::Cyan(0.9f));  
  
    //  
    DrawText(x + size*1.2, y + size*0.3, "km/h", size*0.4, Color::White(0.7f));  
}
```

## 3.4 颜色主题系统

### 3.4.1 黑暗主题

```
void Dashboard::SetDarkTheme() {  
    theme_.primary = Color(0.0f, 0.8f, 1.0f, 0.9f);    //  
    theme_.secondary = Color(1.0f, 1.0f, 1.0f, 0.7f);  //  
    theme_.background = Color(0.1f, 0.1f, 0.1f, 0.8f); //  
    theme_.warning = Color(1.0f, 0.2f, 0.2f, 1.0f);   //  
}
```

### 3.4.2 明亮主题

```
void Dashboard::SetLightTheme() {  
    theme_.primary = Color(0.0f, 0.5f, 0.8f, 0.9f);    //  
    theme_.secondary = Color(0.2f, 0.2f, 0.2f, 0.7f);  //  
    theme_.background = Color(1.0f, 1.0f, 1.0f, 0.7f); //  
    theme_.warning = Color(0.9f, 0.1f, 0.1f, 1.0f);   //  
}
```

## 四、遇到的问题和解决方案

### 4.1 环境配置问题

#### 问题1：网络连接依赖安装失败

现象：在Ubuntu上使用apt install安装依赖时，由于网络问题下载失败。

解决方案：

```
#  
sudo sed -i 's/archive.ubuntu.com/mirrors.aliyun.com/g' /etc/apt/sources.list  
sudo apt update  
  
#  
sudo apt install -y build-essential cmake git \  
    libgl1-mesa-dev libglfw3-dev libglew-dev \  
    libeigen3-dev libopenblas-dev
```

#### 问题2：CMake配置错误

现象：编译时找不到MuJoCo库。

解决方案：在CMakeLists.txt中手动指定路径

```
set(MUJOCO_DIR "/home/username/mujoco_projects/mujoco_mpc")
find_path(MUJOCO_INCLUDE_DIR mujoco/mujoco.h
          PATHS ${MUJOCO_DIR}/include)
```

## 4.2 渲染集成问题

问题3：仪表盘无法显示（渲染顺序错误）

现象：代码编译通过，但运行时仪表盘完全不显示。

原因分析：渲染顺序错误，仪表盘应该在MuJoCo场景渲染之后，但在UI渲染之前。

解决方案：修改 simulate.cc 中的 Render() 函数：

```
void Simulate::Render() {
    // 1. MuJoCo 3D
    mjv_updateScene(m, d, &opt, NULL, &cam, mjCAT_ALL, &scn);
    mjr_render(viewport, &scn, &con);

    // 2. 2D
    mjr_setBuffer(mjFB_WINDOW, &con);

    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glDisable(GL_DEPTH_TEST); //

    //
    if (dashboard_settings.enabled && dashboard_) {
        dashboard_>Render(&con, window_size[0], window_size[1]);
    }

    glEnable(GL_DEPTH_TEST);
    glDisable(GL_BLEND);
    glPopAttrib();

    // 3. MuJoCo UI
    if (ui0_enable) mju_render(&ui0, &uistate, &con);
}
```

关键发现：

- 仪表盘渲染必须在 mju\_render() 之前
- 需要正确设置2D正交投影
- 必须启用透明混合
- 禁用深度测试确保在最上层显示

问题4：字体显示异常

现象：尝试使用复杂字体渲染失败。

解决方案：采用纯图形化设计

```
void Dashboard::DrawDigitalNumber(float x, float y, int number, float size, const Color& color) {
    //
    const bool segment_map[10][7] = {
```

```

        {1,1,1,1,1,1,0}, // 0
        {0,1,1,0,0,0,0}, // 1
        {1,1,0,1,1,0,1}, // 2
        {1,1,1,1,0,0,1}, // 3
        {0,1,1,0,0,1,1}, // 4
        {1,0,1,1,0,1,1}, // 5
        {1,0,1,1,1,1,1}, // 6
        {1,1,1,0,0,0,0}, // 7
        {1,1,1,1,1,1,1}, // 8
        {1,1,1,1,0,1,1} // 9
    };

    //
    for (int i = 0; i < 7; i++) {
        if (segment_map[number][i]) {
            DrawDigitSevenSegment(x, y, i, size, color);
        }
    }
}

```

#### 4.3 性能优化问题

##### 问题5：帧率下降明显

现象：添加仪表盘后，帧率从60FPS下降至30FPS。

优化方案：

```

// 1.
void Dashboard::PrecomputeGeometry() {
    circle_vertices_.clear();
    for (int i = 0; i <= 50; i++) {
        float angle = 2.0f * M_PI * i / 50;
        circle_vertices_.push_back(cos(angle));
        circle_vertices_.push_back(sin(angle));
    }
}

// 2. 批量设置OpenGL状态
void Dashboard::Render(mjrContext* con, int width, int height) {
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glDisable(GL_DEPTH_TEST);

    //
    DrawModernSpeedometer(100, 100, 80, data_.speed_kmh);
    DrawModernTachometer(250, 100, 80, data_.rpm, 8000);
    DrawDigitalSpeed(400, 100, 20, data_.speed_kmh);

    glEnable(GL_DEPTH_TEST);
    glDisable(GL_BLEND);
}

```

优化效果：

- 帧率从30FPS提升至55FPS



- CPU占用率降低35%

## 五、测试与结果

### 5.1 功能测试

测试项	测试方法	预期结果	实际结果	状态
编译运行	执行CMake构建	无错误，正常启动	✓ 编译成功	□
场景加载	加载car_model.xml	显示车辆模型	✓ 车辆正确显示	□
数据获取	移动车辆	速度数据实时更新	✓ 更新频率60Hz	□
速度表	加速/减速	指针平滑移动	✓ 动画平滑	□
转速表	改变速度	RPM同步变化	✓ 红区警告正常	□
数字显示	观察速度	数字实时更新	✓ 显示准确	□
主题切换	调用主题函数	UI颜色变化	✓ 切换即时生效	□
警告系统	触发高温/低油	警告闪烁	✓ 警告效果明显	□

### 5.2 性能测试

#### 5.2.1 帧率对比

测试场景	无仪表盘	有仪表盘(优化前)	有仪表盘(优化后)
简单场景	120 FPS	90 FPS	110 FPS
复杂场景	60 FPS	45 FPS	58 FPS

#### 5.2.2 内存占用

组件	内存占用	说明
Dashboard对象	~5 MB	顶点缓存、动画数据
总增加内存	~5 MB	相对基础内存增加3%

### 5.3 效果展示

视频链接：[https://www.bilibili.com/video/BV1wgBoB4EHL/?vd\\_source=48e56766050425e3578ea1f8b522c0d1](https://www.bilibili.com/video/BV1wgBoB4EHL/?vd_source=48e56766050425e3578ea1f8b522c0d1)

## 六、总结与展望

### 6.1 学习收获

通过本次大作业，我获得了以下重要收获：

技术能力提升：

- 掌握了大型C++项目的二次开发
- 深入理解了物理引擎的工作原理
- 学会了OpenGL 2D渲染技术
- 掌握了实时数据可视化实现

工程实践能力：

- 完整开发流程：环境配置→编码→调试→优化

- 复杂系统调试技巧
- 性能分析和优化经验
- 文档撰写能力

跨学科知识：

- 物理仿真与计算机图形结合
- 实时系统设计与用户体验
- 数据结构与算法应用

## 6.2 项目亮点

现代化UI设计：

- 玻璃效果、平滑动画
- 完整的颜色主题系统
- 七段数码管风格数字显示

系统架构优秀：

- 模块化设计，易于维护
- 与MuJoCo框架无缝集成
- 良好的向后兼容性

性能优化充分：

- 顶点预计算减少实时计算
- 批处理渲染优化
- 数据平滑算法

## 6.3 不足之处

当前局限性：

- 字体渲染依赖简单图形
- 缺少用户交互设置面板
- 功能组件还可以更丰富

改进方向：

- 集成ImGui库，实现设置面板
- 添加更多传感器数据模拟
- 实现数据记录和回放功能

## 6.4 项目意义

- 教育价值：展示物理仿真、计算机图形和实时系统综合应用
- 技术价值：提供现代化仪表盘系统参考实现
- 应用价值：可作为自动驾驶仿真、游戏开发等领域的起点

## 七、参考资料

### 7.1 官方文档

- MuJoCo Documentation: <https://mujoco.readthedocs.io/>
- MuJoCo MPC GitHub: [https://github.com/google-deepmind/mujoco\\_mpc](https://github.com/google-deepmind/mujoco_mpc)
- OpenGL Documentation: <https://www.khronos.org/opengl/>

### 7.2 开发工具

- VSCode: 主要开发环境
- CMake: 构建系统
- Git: 版本控制
- GDB: 调试工具

### 7.3 代码参考

- MuJoCo官方示例代码
- OpenGL图形渲染示例