MuJoCo MPC 汽车仪表盘 - 作业报告

一、项目概述

项目信息

- **学号**：232011191
- **姓名**：郭浩
- **班级**：计科2305
- **完成日期**：2025年12月

1.1 作业背景

随着智能驾驶与自动控制技术的发展，基于物理仿真的车辆控制与状态可视化成为重要的研究与教学内容。MuJoCo（Multi-Joint dynamics with Contact）是一款高性能的物理仿真引擎，广泛应用于机器人与控制领域。本作业基于 MuJoCo 与 mjpc（Model Predictive Control）框架，实现一个简化汽车模型的仪表盘系统，用于直观展示车辆运行状态。

通过本实验，学生可以加深对 MuJoCo 数据结构、物理仿真流程以及实时渲染机制的理解，并掌握仿真数据到可视化结果的完整实现流程。

1.2 实现目标

本项目的主要实现目标如下：

- 基于 MuJoCo 仿真环境搭建简化汽车场景
- 从 MuJoCo 中实时获取车辆位置、速度、加速度等状态数据
- 在仿真场景中渲染 3D 汽车仪表盘（速度表、转速表等）
- 在终端中以原地刷新的方式实时输出车辆状态信息
- 实现基于控制输入的油耗模型，并以百分比形式显示剩余油量
- 保证系统运行稳定，数据更新与渲染同步

1.3 开发环境

- 操作系统：Ubuntu 24.04
- 仿真引擎：MuJoCo
- 控制框架：mjpc
- 编程语言：C++
- 构建工具：CMake
- 编译器：clang12

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

- 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° 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

```
#include
#include
#include
#include

static inline float Deg2Radf(float deg) { return deg * 3.1415926f / 180.0f; }
static inline double Deg2Rad(double deg) { return deg * 3.141592653589793 / 180.0; }

static inline float Clamp01(float v) {
  return std::max(0.0f, std::min(1.0f, v));
}

// 3x3: C = A * B（A/B/C 都是行主序 9 元素）
static inline void Mat3Mul(const double A[9], const double B[9], double C[9]) {
  for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
      C[i * 3 + j] =
        A[i * 3 + 0] * B[0 * 3 + j] +
        A[i * 3 + 1] * B[1 * 3 + j] +
        A[i * 3 + 2] * B[2 * 3 + j];
    }
  }
}

// 绕 Z 轴旋转矩阵（行主序）
static inline void RotZ(double rad, double R[9]) {
  double c = std::cos(rad), s = std::sin(rad);
  R[0] = c;  R[1] = -s; R[2] = 0;
  R[3] = s;  R[4] =  c; R[5] = 0;
  R[6] = 0;  R[7] =  0; R[8] = 1;
}

// -------------------------
// 小工具：往 scene 里塞 geom/label
// -------------------------
static inline bool HasSpace(mjvScene* scene, int need = 1) {
  return scene && (scene->ngeom + need) <= scene->maxgeom;
}

static inline mjvGeom* NewGeom(mjvScene* scene) {
  if (!HasSpace(scene, 1)) return nullptr;
  mjvGeom* g = scene->geoms + scene->ngeom;
  std::memset(g, 0, sizeof(mjvGeom));
  scene->ngeom++;
  return g;
}

static inline void SetMat(mjvGeom* g, const double rot_mat[9]) {
  for (int i = 0; i < 9; ++i) g->mat[i] = static_cast(rot_mat[i]);
}
```

```
static inline void SetRGBA(mjvGeom* g, float r, float gg, float b, float a) {
  g->rgba[0] = r; g->rgba[1] = gg; g->rgba[2] = b; g->rgba[3] = a;
}

static inline void SetPos(mjvGeom* g, float x, float y, float z) {
  g->pos[0] = x; g->pos[1] = y; g->pos[2] = z;
}

static inline void SetSize(mjvGeom* g, float x, float y, float z) {
  g->size[0] = x; g->size[1] = y; g->size[2] = z;
}

static inline void SetLabel(mjvGeom* g, const char* text) {
  std::strncpy(g->label, text, sizeof(g->label) - 1);
  g->label[sizeof(g->label) - 1] = '\0';
}

struct GaugeSpec {
  float arc_radius     = 0.15f;
  int   arc_segments   = 40;

  float tick_radius    = 0.135f;
  float label_radius   = 0.18f;

  float tick_thickness = 0.003f;
  float tick_len_long  = 0.020f;
  float tick_len_short = 0.015f;

  float pointer_half   = 0.11f;   // BOX size[1]（半长）
  float pointer_offset = 0.055f;  // 指针中心离圆心距离

  float hub_radius     = 0.006f;

  float value_scale    = 0.08f;
  float unit_scale     = 0.05f;

  float value_z_offset = 0.02f;
  float unit_z_offset  = -0.06f;

  float arc_rgba[4]    = {0.7f, 0.7f, 0.7f, 0.9f};
  float tick_rgba[4]   = {0.8f, 0.8f, 0.8f, 1.0f};
  float label_rgba[4]  = {0.85f,0.85f,0.85f,1.0f};
  float value_rgba[4]  = {0.92f,0.92f,0.92f,1.0f};
};

// 6 个刻度角（半圆：180 -> 0）
static inline void FillTickAngles(float out_deg[6]) {
  out_deg[0]=180.0f; out_deg[1]=144.0f; out_deg[2]=108.0f;
  out_deg[3]=72.0f;  out_deg[4]=36.0f;  out_deg[5]=0.0f;
}

// -------------------------
// 核心：绘制半圆仪表盘
// -------------------------
static void DrawGauge(
  mjvScene* scene,
```

```
  const double dashboard_rot_mat[9],
  const float center_pos[3],
  float ratio01,
  const int tick_values[6],
  float display_value,
  const char* unit_text,
  float pointer_r, float pointer_g, float pointer_b,
  const GaugeSpec& spec = GaugeSpec()
) {
  if (!scene) return;
  ratio01 = Clamp01(ratio01);

  float tick_deg[6];
  FillTickAngles(tick_deg);

  // 0) 半圆外弧（小段拼起来）
  for (int s = 0; s <= spec.arc_segments; ++s) {
    if (!HasSpace(scene, 1)) break;

    float a_deg = 180.0f - (180.0f * (float)s / (float)spec.arc_segments);
    float rad = Deg2Radf(a_deg);

    float y = center_pos[1] - spec.arc_radius * std::cos(rad);
    float z = center_pos[2] + spec.arc_radius * std::sin(rad);

    mjvGeom* g = NewGeom(scene);
    if (!g) break;
    g->type = mjGEOM_BOX;
    SetSize(g, 0.0025f, 0.0045f, 0.0025f);
    SetPos(g, center_pos[0], y, z);
    SetMat(g, dashboard_rot_mat);
    SetRGBA(g, spec.arc_rgba[0], spec.arc_rgba[1], spec.arc_rgba[2],
    spec.arc_rgba[3]);

  }

  // 1) 刻度线 + 数字
  for (int i = 0; i < 6; ++i) {
    if (!HasSpace(scene, 2)) break; // tick + label

    float rad = Deg2Radf(tick_deg[i]);
    float tick_len = (i == 0 || i == 5 || i == 2) ? spec.tick_len_long :
    spec.tick_len_short;

    float tick_y = center_pos[1] - spec.tick_radius * std::cos(rad);
    float tick_z = center_pos[2] + spec.tick_radius * std::sin(rad);

    // tick
    {
      mjvGeom* g = NewGeom(scene);
      if (!g) break;
      g->type = mjGEOM_BOX;
      SetSize(g, spec.tick_thickness, tick_len, spec.tick_thickness);
      SetPos(g, center_pos[0], tick_y, tick_z);
```

```
  SetMat(g, dashboard_rot_mat);
  SetRGBA(g, spec.tick_rgba[0], spec.tick_rgba[1], spec.tick_rgba[2],
 spec.tick_rgba[3]);
 }

 // label
 {
  mjvGeom* t = NewGeom(scene);
  if (!t) break;
  t->type = mjGEOM_LABEL;
  SetSize(t, 0.05f, 0.05f, 0.05f);

  float label_y = center_pos[1] - spec.label_radius * std::cos(rad);
  float label_z = center_pos[2] + spec.label_radius * std::sin(rad);
  SetPos(t, center_pos[0], label_y, label_z);

  SetRGBA(t, spec.label_rgba[0], spec.label_rgba[1], spec.label_rgba[2],
 spec.label_rgba[3]);

  char buf[32];
  std::snprintf(buf, sizeof(buf), "%d", tick_values[i]);
  SetLabel(t, buf);
 }
```

}

// 2) 指针（半圆：180 -> 0)
if (HasSpace(scene, 1)) {
 float angle_deg = 180.0f - 180.0f * ratio01;
 float rad = Deg2Radf(angle_deg);

```
 float py = center_pos[1] - spec.pointer_offset * std::cos(rad);
 float pz = center_pos[2] + spec.pointer_offset * std::sin(rad);

 mjvGeom* g = NewGeom(scene);
 if (g) {
  g->type = mjGEOM_BOX;
  SetSize(g, 0.004f, spec.pointer_half, 0.003f);
  SetPos(g, center_pos[0], py, pz);

  // 指针自身绕 z 旋转（相对仪表盘平面）
  double Rpointer[9];
  RotZ(Deg2Rad((double)angle_deg - 90.0), Rpointer);

  double final_mat[9];
  Mat3Mul(dashboard_rot_mat, Rpointer, final_mat); // 先仪表盘面，再叠加指针角
  SetMat(g, final_mat);

  SetRGBA(g, pointer_r, pointer_g, pointer_b, 1.0f);
 }
```

}

```
  // 3) 中心点
  if (HasSpace(scene, 1)) {
    mjvGeom* g = NewGeom(scene);
    if (g) {
      g->type = mjGEOM_SPHERE;
      SetSize(g, spec.hub_radius, spec.hub_radius, spec.hub_radius);
      SetPos(g, center_pos[0], center_pos[1], center_pos[2]);
      SetMat(g, dashboard_rot_mat);
      SetRGBA(g, 0.8f, 0.8f, 0.8f, 1.0f);
    }
  }

  // 4) 数字（中心偏上）
  if (HasSpace(scene, 1)) {
    mjvGeom* g = NewGeom(scene);
    if (g) {
      g->type = mjGEOM_LABEL;
      SetSize(g, spec.value_scale, spec.value_scale, spec.value_scale);
      SetPos(g, center_pos[0], center_pos[1], center_pos[2] + spec.value_z_offset);
      SetRGBA(g, spec.value_rgba[0], spec.value_rgba[1], spec.value_rgba[2], spec.value_rgba[3]);

      char buf[64];
      std::snprintf(buf, sizeof(buf), "%.0f", display_value);
      SetLabel(g, buf);
    }

  }

  // 5) 单位
  if (HasSpace(scene, 1)) {
    mjvGeom* g = NewGeom(scene);
    if (g) {
      g->type = mjGEOM_LABEL;
      SetSize(g, spec.unit_scale, spec.unit_scale, spec.unit_scale);
      SetPos(g, center_pos[0], center_pos[1], center_pos[2] + spec.unit_z_offset);
      SetRGBA(g, spec.label_rgba[0], spec.label_rgba[1], spec.label_rgba[2], spec.label_rgba[3]);
      SetLabel(g, unit_text);
    }
  }
}

{
  const int speed_ticks[6] = {0, 2, 4, 6, 8, 10};
  const int rpm_ticks[6]   = {0, 1600, 3200, 4800, 6400, 8000};

  // 左：速度
  DrawGauge(scene, dashboard_rot_mat,
            speed_dashboard_pos,
            speed_ratio,
            speed_ticks,
            static_cast(speed_kmh),
```

```
        "km/h",
        1.0f, 0.0f, 0.0f);

 // 右：转速
 DrawGauge(scene, dashboard_rot_mat,
        rpm_dashboard_pos,
        rpm_ratio,
        rpm_ticks,
        rpm_value,
        "RPM",
        1.0f, 1.0f, 0.0f);

 // 中间 Fuel
 if (scene && scene->ngeom < scene->maxgeom) {
   mjvGeom* mid = scene->geoms + scene->ngeom;
   std::memset(mid, 0, sizeof(mjvGeom));
   mid->type = mjGEOM_LABEL;
   mid->size[0] = mid->size[1] = mid->size[2] = 0.06f;
```

```
  mid->pos[0] = base_pos[0];
  mid->pos[1] = base_pos[1];
  mid->pos[2] = base_pos[2] + 0.06f;

  mid->rgba[0] = 0.95f;
  mid->rgba[1] = 0.95f;
  mid->rgba[2] = 0.95f;
  mid->rgba[3] = 1.0f;

  char mid_label[64];
  std::snprintf(mid_label, sizeof(mid_label), "Fuel %.0f%%", fuel_percent);
  std::strncpy(mid->label, mid_label, sizeof(mid->label) - 1);
  mid->label[sizeof(mid->label) - 1] = '\0';

  scene->ngeom++;
```

```
 }
}
```

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

代码片段

```
// ===== 转速盘比例（模拟）=====
#include
#include
```

```
// 角度 -> 弧度
static inline double Deg2Rad(double deg) {
  return deg * M_PI / 180.0;
}

// 限幅到 [lo, hi]
static inline float Clamp(float v, float lo, float hi) {
  return std::max(lo, std::min(v, hi));
}

// 3x3 矩阵乘法：C = A * B（行主序）
static inline void Mat3Mul(const double A[9], const double B[9], double C[9]) {
  for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
      C[i * 3 + j] =
          A[i * 3 + 0] * B[0 * 3 + j] +
          A[i * 3 + 1] * B[1 * 3 + j] +
          A[i * 3 + 2] * B[2 * 3 + j];
    }
  }
}

// 绕 X 轴旋转矩阵（行主序）
static inline void RotX(double rad, double R[9]) {
  double c = std::cos(rad), s = std::sin(rad);
  R[0] = 1; R[1] = 0;  R[2] = 0;
  R[3] = 0; R[4] = c;  R[5] = -s;
  R[6] = 0; R[7] = s;  R[8] = c;
}

// 绕 Z 轴旋转矩阵（行主序）
static inline void RotZ(double rad, double R[9]) {
  double c = std::cos(rad), s = std::sin(rad);
  R[0] = c;  R[1] = -s; R[2] = 0;
  R[3] = s;  R[4] = c;  R[5] = 0;
  R[6] = 0;  R[7] = 0;  R[8] = 1;
}

// ===== 转速盘比例（模拟）=====
const float max_rpm = 8000.0f;
float rpm_ratio = static_cast(speed_m / max_speed_ref);
rpm_ratio = Clamp(rpm_ratio, 0.0f, 1.0f);
float rpm_value = rpm_ratio * max_rpm;

// ===== 仪表盘旋转（立起来 + 顺时针90度）=====
double Rx[9], Rz[9];
RotX(Deg2Rad(90.0), Rx);
RotZ(Deg2Rad(-90.0), Rz);

double dashboard_rot_mat[9];
Mat3Mul(Rz, Rx, dashboard_rot_mat);  // 先 X 后 Z：R = Rz * Rx
```

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、遇到的问题和解决方案

问题1

- **现象**：终端输出频繁刷屏，难以阅读
- **原因**：每帧使用换行符输出数据
- **解决**：使用回车符 `\r` 并强制刷新输出缓冲区，实现原地刷新

问题2

- **现象**：编译时报变量重复定义错误
- **原因**：在同一作用域中多次定义相同变量
- **解决**：统一变量定义位置，仅在首次使用时定义

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

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

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
- 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