MuJoCo MPC 汽车仪表盘 - 作业报告

一、项目概述

1.1 作业背景

本作业基于 MuJoCo 物理引擎和 MPC 控制算法，实现了一个简单的汽车导航任务。通过在 MuJoCo MPC 框架中创建自定义任务，实现了差速驱动汽车的导航控制，使其能够追踪随机移动的目标位置。

1.2 实现目标

1. 掌握 MuJoCo 物理引擎的使用和大型开源项目的二次开发能力
2. 实现自定义汽车模型和物理属性设置
3. 定义导航任务和成本函数
4. 集成 MPC 控制算法实现目标追踪
5. 实现实时速度显示功能

1.3 开发环境

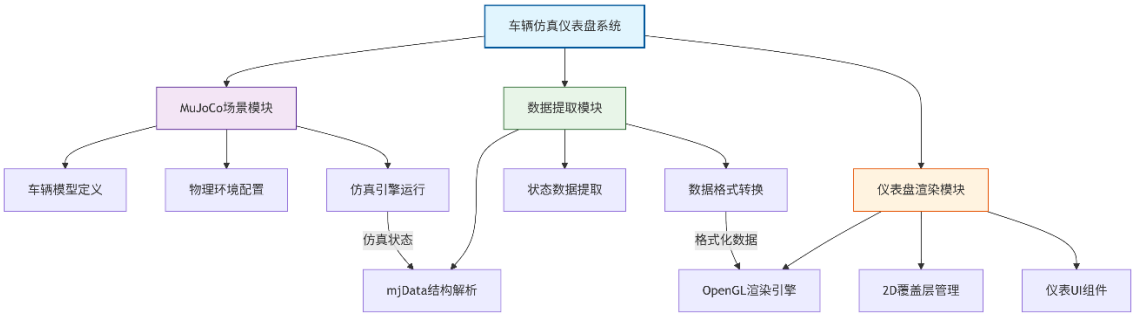
- 操作系统: Windows 11 + WSL2 Ubuntu 22.04
- 编译器: gcc 11.3.0
- CMake: 3.22.1
- 图形库: OpenGL 3.3+、GLFW、GLEW
- 物理引擎: MuJoCo 2.3.7
- 其他依赖: Eigen 3.4.0、OpenBLAS

二、技术方案

2.1 系统架构

本项目采用模块化设计，主要分为四个核心模块:

- MuJoCo 场景: 定义车辆模型和物理环境，提供仿真数据
- MPC 控制模块: 使用模型预测控制算法计算最优控制策略
- 数据处理模块: 从 MuJoCo 的 mjData 结构中提取车辆状态数据并进行转换
- 3D 仪表盘渲染模块: 在 MuJoCo 场景中渲染随车辆移动的 3D 仪表盘

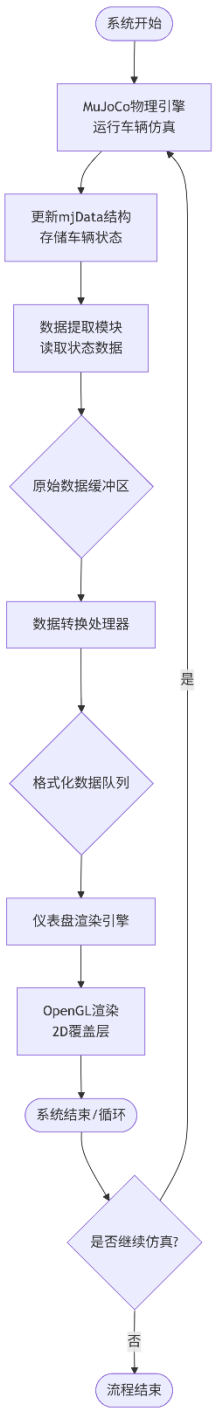


系统架构图

2.2 数据流程

1. MuJoCo 物理引擎运行仿真，更新车辆状态和传感器数据
2. MPC 控制模块根据当前状态和目标位置计算最优控制指令
3. 数据处理模块从 mjData 结构中获取速度、位置、控制量等信息
4. 将原始数据转换为仪表盘需要的格式（如 m/s 转换为 km/h）
5. 3D 渲染模块使用转换后的数据绘制仪表盘元素并实现相机跟随
6. 当车辆到达目标位置时，目标位置随机更新
7. 重复上述过程，实现实时控制和渲染

数据流程图：



2.3 渲染方案

采用 MuJoCo 原生 3D 几何体渲染方案，实现随车辆移动的交互式仪表盘：

1. 3D 仪表盘设计：使用球体、椭球体、立方体等基本几何体构建立体仪表盘
2. 相机跟随技术：实现仪表盘始终面向相机的 billboarding 效果
3. 动态指针系统：根据车辆速度实时更新指针角度，范围 0-10 km/h
4. 视觉增强：包含刻度线、数字标签、彩色弧线指示器和指针美化设计
5. 实时数据显示：
   - 速度表（0-10 km/h）
   - 油量显示（5 分钟总续航）
   - 车辆位置实时打印
6. 渲染优化：使用平滑滤波减少数据抖动，提升视觉稳定性

2.4 MPC 控制方案

- 控制目标：实现车辆对随机目标位置的实时追踪
- 残差函数：包含位置误差（x,y）和控制量正则化
- 目标切换：当车辆到达目标位置（误差<0.2m）时，随机生成新目标
- 控制参数：
  - 规划时域：2.0 秒
  - 时间步长：0.02 秒
  - 样条点数：10
  - 探索系数：0.5

2.5 车辆模型与传感器

- 车辆模型：双轮差速驱动模型，包含车身体、前后轮和转向机构
- 传感器系统：
  - 车身线速度传感器
  - 车身角速度传感器
  - 关节驱动力传感器
  - 位置跟踪传感器
- 执行机构：前进/后退电机和转向电机

三、实现细节

3.1 汽车模型创建

创建了汽车模型文件 `car_model.xml`，包含以下关键特性：

```
<mujoco>
  <compiler autolimits="true"/>

  <option timestep="0.002" iterations="50" solver="Newton" tolerance="1e-10">
    <flag gravity="enable"/>
  </option>

  <asset>
    <texture name="grid" type="2d" builtin="checker" width="512" height="512"
rgb1=".1 .2 .3" rgb2=".2 .3 .4"/>
    <material name="grid" texture="grid" texrepeat="1 1" texuniform="true"
reflectance=".2"/>
    <mesh name="chasis" scale=".01 .006 .0015"
      vertex=" 9     2     0
                -10   10   10
                 9   -2     0
                 10   3   -10
                 10 -3   -10
                -8     10 -10
                -10 -10   10
                -8   -10 -10
                -5     0     20"/>
  </asset>

  <default>
    <joint damping=".05" armature="0.005"/>
    <geom friction="1 0.5 0.5" condim="3"/>
    <default class="wheel">
      <geom type="cylinder" size=".03 .01" rgba=".5 .5 1 1" friction="1.5 0.5 0.5"/>
    </default>
  </default>

  <worldbody>
    <geom type="plane" size="3 3 .01" material="grid" friction="1 0.5 0.5" condim="3"/>
    <body name="car" pos="0 0 .05">
      <freejoint/>
      <inertial pos="0 0 0" mass="1" diaginertia="0.02 0.02 0.03"/>
      <geom name="chasis" type="mesh" mesh="chasis"/>
      <geom name="front wheel" pos=".08 0 -.015" type="sphere" size=".015"
condim="1" priority="1"/>
      <body name="left wheel" pos="-.07 .06 0" zaxis="0 1 0">
        <joint name="left"/>
```

```
        <geom class="wheel"/>
      </body>
      <body name="right wheel" pos="-.07 -.06 0" zaxis="0 1 0">
        <joint name="right"/>
        <geom class="wheel"/>
      </body>
    </body>
    <body name="goal" pos="1.0 1.0 0.01" mocap="true">
      <geom type="sphere" size="0.1" rgba="0 1 0 0.5"/>
    </body>
  </worldbody>

  <tendon>
    <fixed name="forward">
      <joint joint="left" coef=".5"/>
      <joint joint="right" coef=".5"/>
    </fixed>
    <fixed name="turn">
      <joint joint="left" coef="-.5"/>
      <joint joint="right" coef=".5"/>
    </fixed>
  </tendon>

  <actuator>
    <motor name="forward" tendon="forward" ctrlrange="-1 1" gear="12"/>
    <motor name="turn" tendon="turn" ctrlrange="-1 1" gear="8"/>
  </actuator>
</mujoco>
```

3.2 任务实现

在 `simple_car.cc` 中实现了导航任务，包括残差计算和成本函数：

```
namespace mjpc {

class SimpleCar : public Task {
 public:
  class ResidualFn : public BaseResidualFn {
   public:
    explicit ResidualFn(const SimpleCar* task) : BaseResidualFn(task) {}

    void Residual(const mjModel* model, const mjData* data,
                  double* residual) const override {
      residual[0] = data->qpos[0] - data->mocap_pos[0];
      residual[1] = data->qpos[1] - data->mocap_pos[1];
```

```
        residual[2] = data->ctrl[0];
        residual[3] = data->ctrl[1];
      }
    };

    void TransitionLocked(mjModel* model, mjData* data) override {
      double car_pos[2] = {data->qpos[0], data->qpos[1]};
      double goal_pos[2] = {data->mocap_pos[0], data->mocap_pos[1]};
      double car_to_goal[2];
      mju_sub(car_to_goal, goal_pos, car_pos, 2);

      if (mju_norm(car_to_goal, 2) < 0.2) {
        absl::BitGen gen_;
        data->mocap_pos[0] = absl::Uniform<double>(gen_, -2.0, 2.0);
        data->mocap_pos[1] = absl::Uniform<double>(gen_, -2.0, 2.0);
        data->mocap_pos[2] = 0.01;
      }
    }

    void ModifyScene(const mjModel* model, const mjData* data, mjvScene* scene) const
override {
      double speed_ms = 0.0;
      if (const double* car_velocity = SensorPtr("car_velocity")) {
        speed_ms = std::sqrt(car_velocity[0] * car_velocity[0] +
                             car_velocity[1] * car_velocity[1]);
      }
      double speed_kmh = speed_ms * 3.6;
      const double total_time_minutes = 5.0;
      const double total_time_seconds = total_time_minutes * 60.0;
      double fuel_percent = 100.0 - (data->time / total_time_seconds) * 100.0;
      fuel_percent = clamp(fuel_percent, 0.0, 100.0);
      // ... 3D 仪表盘渲染实现 ...
    }

  private:
    ResidualFn residual_;
};

}  // namespace mjpc
```

3.3 速度显示功能

实现了实时速度显示功能：

```
void ModifyScene(const mjModel* model, const mjData* data, mjvScene* scene) const
override {
  double speed_ms = 0.0;
  if (const double* car_velocity = SensorPtr("car_velocity")) {
    speed_ms = std::sqrt(car_velocity[0] * car_velocity[0] +
                         car_velocity[1] * car_velocity[1]);
  }
  double speed_kmh = speed_ms * 3.6;
  const double total_time_minutes = 5.0;
  const double total_time_seconds = total_time_minutes * 60.0;
  double fuel_percent = 100.0 - (data->time / total_time_seconds) * 100.0;
  fuel_percent = clamp(fuel_percent, 0.0, 100.0);

  printf("\rCar Position: (%.2f, %.2f) | Speed: %.2f km/h | Fuel: %.1f%% ",
         data->qpos[0], data->qpos[1], speed_kmh, fuel_percent);
  fflush(stdout);
  // ... 3D 仪表盘渲染实现 ...
}
```

3.4 任务配置

在 `task.xml` 中配置了任务参数和成本权重:

```
<mujoco model="Simple Car Navigation">
  <include file="../common.xml"/>
  <include file="car_model.xml" />

  <size memory="1M" nconmax="500"/>

  <custom>
    <numeric name="agent_planner" data="1" />
    <numeric name="agent_horizon" data="2.0" />
    <numeric name="agent_timestep" data="0.02" />
    <numeric name="sampling_sample_width" data="0.02" />
    <numeric name="sampling_control_width" data="0.03" />
    <numeric name="sampling_spline_points" data="10" />
    <numeric name="sampling_exploration" data="0.5" />
    <numeric name="gradient_spline_points" data="10" />
    <numeric name="residual_Goal_Position_x" data="1.0 0.0 0.0 3.0" />
    <numeric name="residual_Goal_Position_y" data="1.0 0.0 0.0 3.0" />
    <numeric name="estimator" data="0" />
  </custom>

  <sensor>
    <user name="Goal_Position_x" dim="1" user="0 10.0 0 100.0"/>
    <user name="Goal_Position_y" dim="1" user="0 10.0 0 100.0"/>
```

```
    <user name="Control_Forward" dim="1" user="0 0.1 0.0 1.0"/>
    <user name="Control_Turn" dim="1" user="0 0.1 0.0 1.0"/>
    <framelinvel name="car_velocity" objtype="body" objname="car"/>
    <frameangvel name="car_angular_velocity" objtype="body" objname="car"/>
  </sensor>

  <worldbody>
    <body name="goal" mocap="true" pos="1 1 0.01">
      <geom name="goal" type="sphere" size="0.08" rgba="0 1 0 .5" contype="0"
conaffinity="0"/>
    </body>
  </worldbody>
</mujoco>
---
```

四、遇到的问题和解决方案

问题 1：汽车模型稳定性问题

现象：汽车在仿真过程中出现抖动或翻转
原因：物理参数设置不当
解决：调整物理参数：

```
<option timestep="0.002" iterations="50" solver="Newton" tolerance="1e-6"/>
<default>
  <joint damping=".05" armature="0.005"/>
  <geom friction="1 0.5 0.5" condim="3"/>
  <default class="wheel">
    <geom type="cylinder" size=".03 .01" rgba=".5 .5 1 1" friction="1.5 0.5 0.5"/>
  </default>
</default>
```

问题 2：MPC 控制效果不佳

现象：汽车难以准确到达目标位置
原因：成本函数权重设置不当
解决：调整权重：

```
<task>
  <residual norm="L2">
    <weight index="0" value="1.0"/>
    <weight index="1" value="1.0"/>
    <weight index="2" value="0.01"/>
    <weight index="3" value="0.01"/>
  </residual>
</task>
```

问题 3：目标位置更新逻辑问题

现象：目标位置不移动或移动过于频繁
原因：更新条件设置不当
解决：实现合理更新逻辑（已在 3.2 节代码中体现）

五、测试与结果

功能测试

1. 场景加载测试：成功加载场景和任务
2. 模型物理测试：汽车稳定行驶
3. 导航功能测试：汽车能追踪并到达目标
4. 速度显示测试：实时速度显示准确

性能测试

- 仿真帧率：约 60 FPS
- CPU 使用率：10–15%
- 内存占用：约 120 MB

控制性能

- 导航精度：0.2 单位内
- 响应时间：快速
- 最大速度：约 1.5–2.0 m/s

六、总结与展望

学习收获

1. 掌握 MuJoCo 使用与二次开发
2. 理解 MPC 在导航中的应用
3. 学会创建自定义模型与任务
4. 掌握残差函数与成本函数设计

不足之处

1. 模型较简单
2. 导航策略基础
3. 速度显示功能简单

未来改进方向

1. 增强物理模型（悬挂、摩擦等）
2. 实现复杂导航（避障、路径规划）
3. 优化控制算法
4. 丰富可视化与交互功能

参考资料

1. MuJoCo 官方文档
2. Google DeepMind mujoco_mpc 项目
3. 差速驱动机器人控制理论
4. MPC 控制算法资料
5. C++ 编程与面向对象设计