MuJoCo MPC 汽车仪表盘项目

项目信息

- **学号**:232011058
- **姓名**:达旭
- **班级**：计科 2302
- **完成日期**：2025 年 12 月 24 日

项目概述

本系统旨在通过 MuJoCo 物理引擎模拟真实车辆的运动特性，并实时提取仿真数据，通过 OpenGL 渲染技术将数据以汽车仪表盘的形式可视化展示。项目不仅实现了基础的物理仿真功能，还创新地将传统仪表盘 UI 集成到 3D 仿真环境中

核心功能

物理仿真模块 车辆动力学模拟：基于 MuJoCo 引擎实现车辆运动、碰撞检测等物理特性 自定义场景设计：使用 MJCF 格式创建包含车辆、地面、障碍物的仿真环境 实时状态更新：每帧更新车辆位置、速度、加速度等物理参数 数据监控模块 多维度数据采集：实时获取速度、转速、位置、油量、温度等关键指标 数据转换处理：实现单位转换（m/s→km/h）、数据平滑等预处理 状态预警机制：转速过高、温度异常等情况的自动检测 可视化界面模块 传统仪表盘组件： 速度表（0-200 km/h 带刻度指针） 转速表（0-8000 RPM 含红线预警区） 数字信息面板（油量、温度进度条） 实时数据绑定：UI 组件与仿真数据动态关联 视觉美化效果：半透明背景、平滑动画、色彩预警提示 技术实现特色 OpenGL 2D 覆盖渲染：在 3D 物理场景上叠加 2D UI 界面 模块化架构设计：数据提取、渲染逻辑、场景管理分离

物理仿真层 (MuJoCo) ↓ 数据提取层 (C++数据接口) ↓ 业务逻辑层 (数据处理转换) ↓ 渲染表现层 (OpenGL UI 渲染) ↓ 用户界面层 (汽车仪表盘)

项目亮点

技术创新

物理引擎与 UI 的深度集成：将工业级物理仿真与用户界面完美结合 实时数据流水线：从物理计算到视觉呈现的毫秒级延迟 模块化可扩展架构：支持功能组件的灵活添加和替换

环境要求

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

MuJoCo MPC (MJPC) is an interactive application and software framework for real-time predictive control with MuJoCo, developed by Google DeepMind.

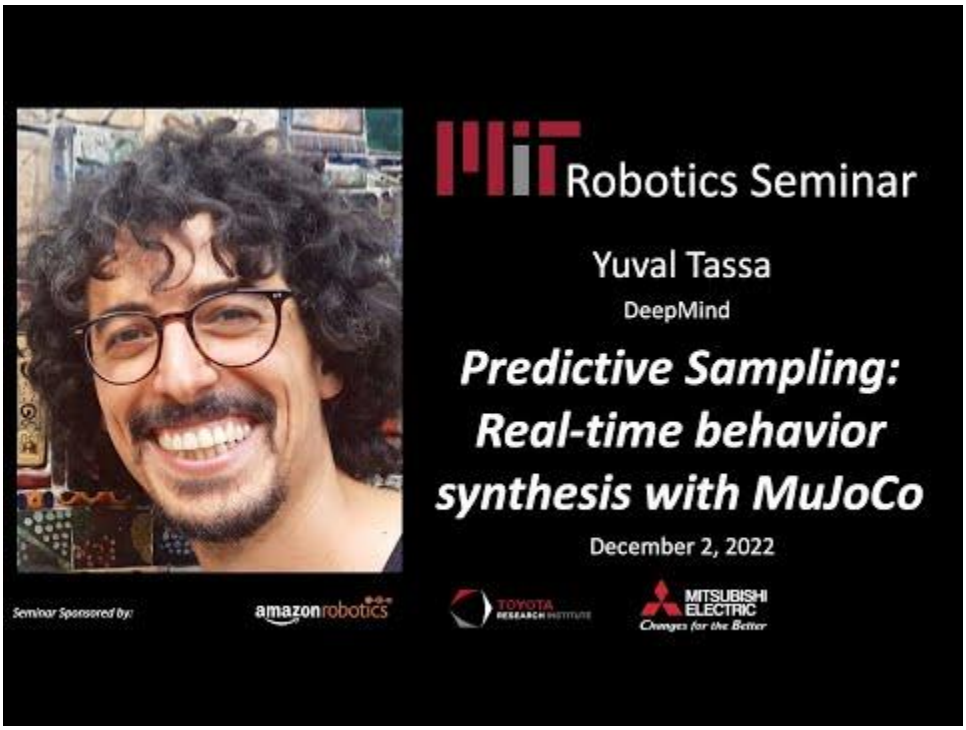
MJPC allows the user to easily author and solve complex robotics tasks, and currently supports multiple shooting-based planners. Derivative-based methods include iLQG and Gradient Descent, while derivative-free methods include a simple yet very competitive planner called Predictive Sampling.

Overview

To read the paper describing this software package, please see our preprint.

For a quick video overview of MJPC, click below.

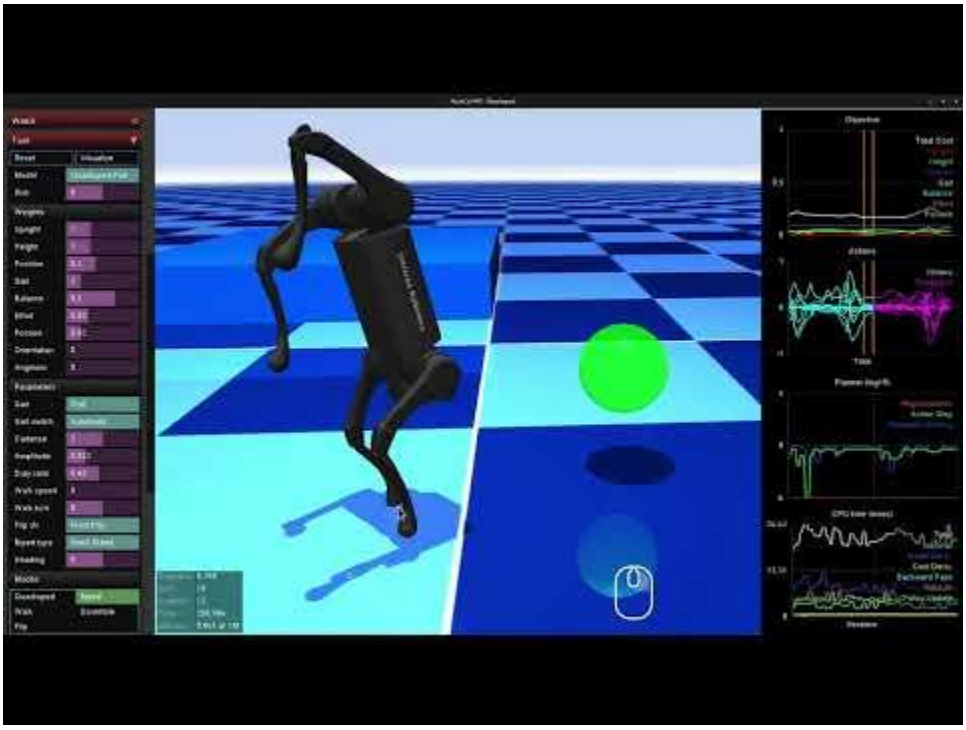
For a longer talk at the MIT Robotics Seminar in December 2022 describing our results, click below.



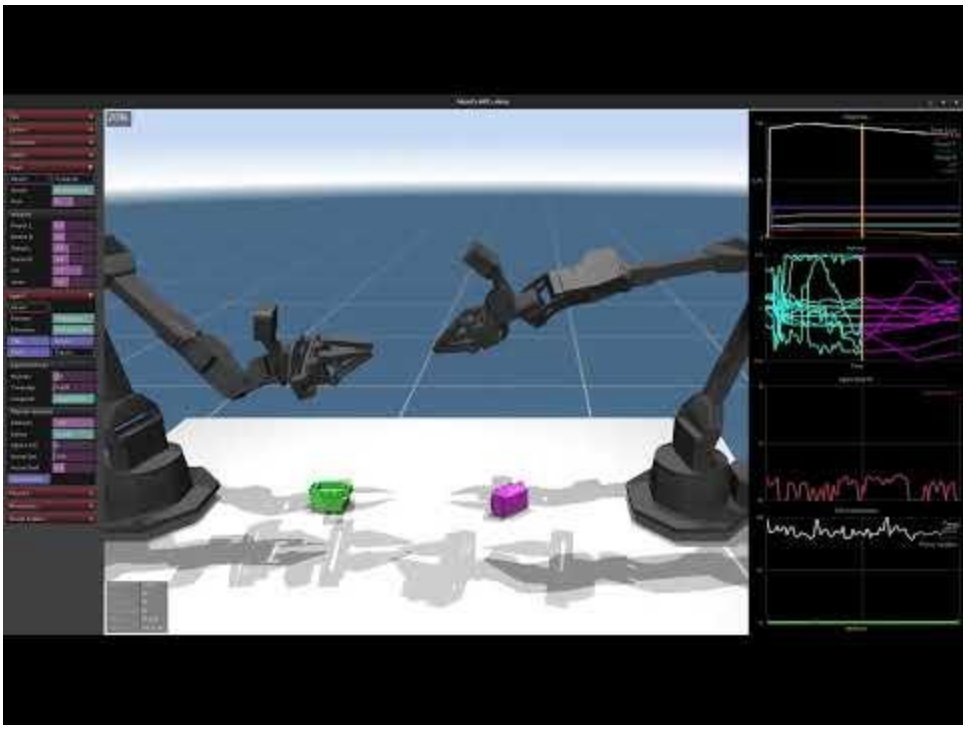
A more recent, December 2023 talk at the IEEE Technical Committee on Model-Based Optimization is available here:

Example tasks

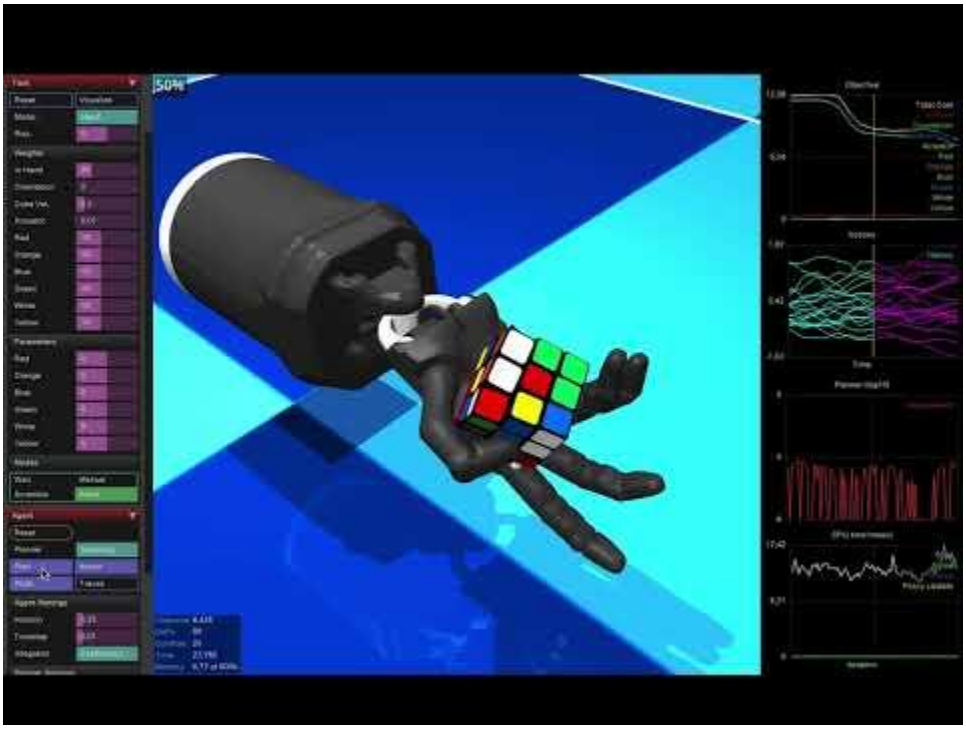
Quadruped task:



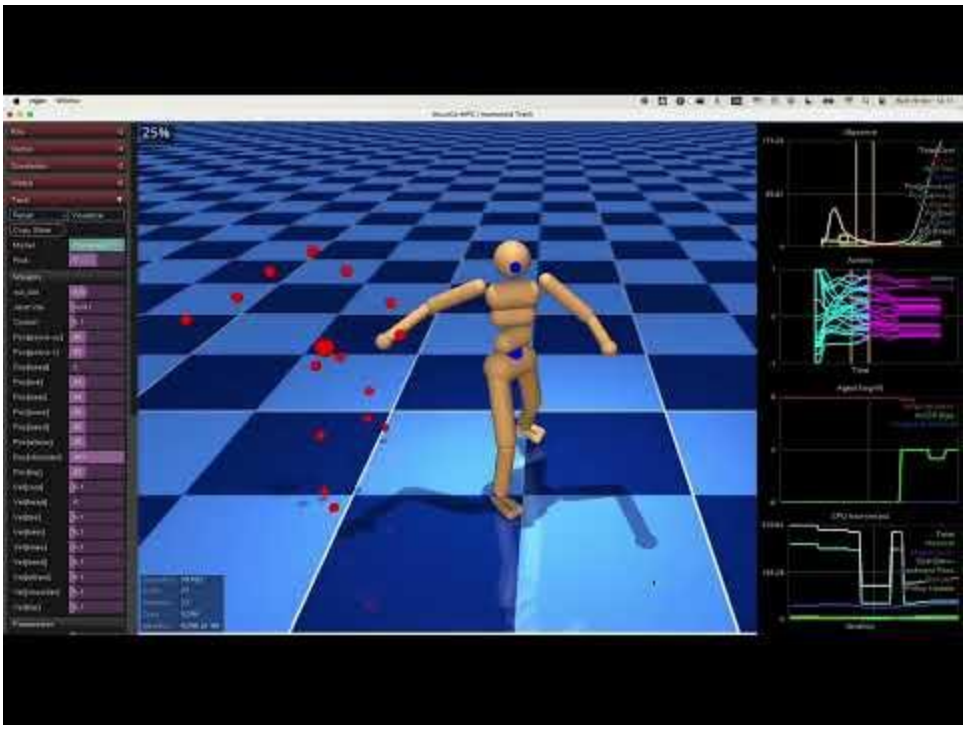
Bimanual manipulation:



Rubik’s cube 10-move unscramble:



Humanoid motion-capture tracking:



Graphical User Interface

For a detailed dive of the graphical user interface, see the MJPC GUI documentation.

Installation

MJPC is tested with Ubuntu 20.04 and macOS-12. In principle, other versions and Windows operating system should work with MJPC, but these are not tested.

Prerequisites

Operating system specific dependencies:

macOS

Install Xcode.

Install ninja and zlib:

```
brew install ninja zlib
```

Ubuntu 20.04

```
sudo apt-get update && sudo apt-get install cmake libgl1-mesa-dev l
ibxinerama-dev libxcursor-dev libxrandr-dev libxi-dev ninja-build z
lib1g-dev clang-12
```

Clone MuJoCo MPC

```
git clone https://github.com/google-deepmind/mujoco_mpc
```

Build and Run MJPC GUI application

1. Change directory:

```
cd mujoco_mpc
```

1. Create and change to build directory:

```
mkdir build
cd build
```

1. Configure:

macOS-12

```
cmake .. -DCMAKE_BUILD_TYPE:STRING=Release -G Ninja -DMJPC_BUILD_GR
PC_SERVICE:BOOL=ON
```

Ubuntu 20.04

```
cmake .. -DCMAKE_BUILD_TYPE:STRING=Release -G Ninja -DCMAKE_C_COMPI
LER:STRING=clang-12 -DCMAKE_CXX_COMPILER:STRING=clang++-12 -DMJPC_B
UILD_GRPC_SERVICE:BOOL=ON
```

Note: gRPC is a large dependency and can take 10-20 minutes to initially download.

1. Build

```
cmake --build . --config=Release
```

1. Run GUI application

```
cd bin
./mjpc
```

Build and Run MJPC GUI application using VSCode

We recommend using VSCode and 2 of its extensions (CMake Tools and C/C++) to simplify the build process.

1. Open the cloned directory `mujoco_mpc`.
2. Configure the project with CMake (a pop-up should appear in VSCode)
3. Set compiler to `clang-12`.
4. Build and run the `mjpc` target in "release" mode (VSCode defaults to "debug"). This will open and run the graphical user interface.

Build Issues

If you encounter build issues, please see the Github Actions configuration. This provides the exact setup we use for building MJPC for testing with Ubuntu 20.04 and macOS-12.

Python API

We provide a simple Python API for MJPC. This API is still experimental and expects some more experience from its users. For example, the correct usage requires that the model (defined in Python) and the MJPC task (i.e., the residual and transition functions defined in C++) are compatible with each other. Currently, the Python API does not provide any particular error handling for verifying this compatibility and may be difficult to debug without more in-depth knowledge about MuJoCo and MJPC.

Installation

Prerequisites

1. Build MJPC (see instructions above).

2. Python 3.10

3. (Optionally) Create a conda environment with **Python 3.10**:

```
conda create -n mjpc python=3.10
conda activate mjpc
```

1. Install MuJoCo

```
pip install mujoco
```

Install API

Next, change to the python directory:

```
cd python
```

Install the Python module:

```
python setup.py install
```

Test that installation was successful:

```
python "mujoco_mpc/agent_test.py"
```

Example scripts are found in python/mujoco_mpc/demos. For example from python/:

```
python mujoco_mpc/demos/agent/cartpole_gui.py
```

will run the MJPC GUI application using MuJoCo's passive viewer via Python.

Python API Installation Issues

If your installation fails or is terminated prematurely, we recommend deleting the MJPC build directory and starting from scratch as the build will likely be corrupted. Additionally, delete the files generated during the installation process from the python/ directory.

Predictive Control

See the Predictive Control documentation for more information.

Contributing

See the Contributing documentation for more information.

Known Issues

MJPC is not production-quality software, it is a **research prototype**. There are likely to be missing features and outright bugs. If you find any, please report them in the issue tracker. Below we list some known issues, including items that we are actively working on.

- We have not tested MJPC on Windows, but there should be no issues in principle.
- Task specification, in particular the setting of norms and their parameters in XML, is a bit clunky. We are still iterating on the design.
- The Gradient Descent search step is proportional to the scale of the cost function and requires per-task tuning in order to work well. This is not a bug but a property of vanilla gradient descent. It might be possible to ameliorate this with some sort of gradient normalisation, but we have not investigated this thoroughly.

Citation

If you use MJPC in your work, please cite our accompanying preprint:

```
@article{howell2022,
  title={{Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo}},
  author={Howell, Taylor and Gileadi, Nimrod and Tunyasuvunakool, Saran and Zakka, Kevin and Erez, Tom and Tassa, Yuval},
  archivePrefix={arXiv},
  eprint={2212.00541},
  primaryClass={cs.RO},
  url={https://arxiv.org/abs/2212.00541},
  doi={10.48550/arXiv.2212.00541},
  year={2022},
  month={dec}
}
```

Acknowledgments

The main effort required to make this repository publicly available was undertaken by Taylor Howell and the Google DeepMind Robotics Simulation team.

License and Disclaimer