Robot Dynamics and Control

- Dynamics Simulation using MuJoCo

2025.05.19 (Mon) Part.1 : Introduction & Installation
2025.05.21 (Wed) Part.2 : Programming Practice

TA: Jeongwoo Hong
email: jwhong1209@gmail.com

Today's Contents

1. Introduction
   - Why Dynamics Simulation?
   - What is MuJoCo?

2. Key Components of MuJoCo
   - Interactive GUI (Graphical User Interface)
   - Robot Model Description: MJCF File
   - Programming API

3. Getting Started
   - Development Environment Set Up
   - Quick Review of C++ Build Process
   - Installation
   - Practice Template
   - How it works

4. Wrap-ups

Why Dynamics Simulation?

© Boston Dynamics. Spot

"Hardware is Hard"

- **Mechanics / Electronics** Issues
- **Uncertainties** (Modeling, Friction, Environment)
- **Potential Physical risk**
- Require lots of **Time, Effort, and Money**

Movie: The Matrix

- **Fast Prototyping**
- **Zero Physical Risk**
- **Fully Reproducible Environment**

What is MuJoCo?

MuJoCo (Multi – Joint dynamics with Contact)

- Physics engine for model-based control
- Created by Emanuel with help from Tom and Yuval (Todorove et al. 2012)
- **Open-sourced by Google Deepmind** in May 2022
- Actively supported and developed
- Full featured, **well-documented**
- Key Features
  - *Generalized coordinates* combined with *modern contact dynamics*
  - Choice of Euler, implicit, and Runge-Kutta (RK4) *numerical integrators*
  - *Intuitive XML model format* and built-in model compiler
  - *Cross-platform GUI* with interactive 3D visualization in OpenGL

MuJoCo

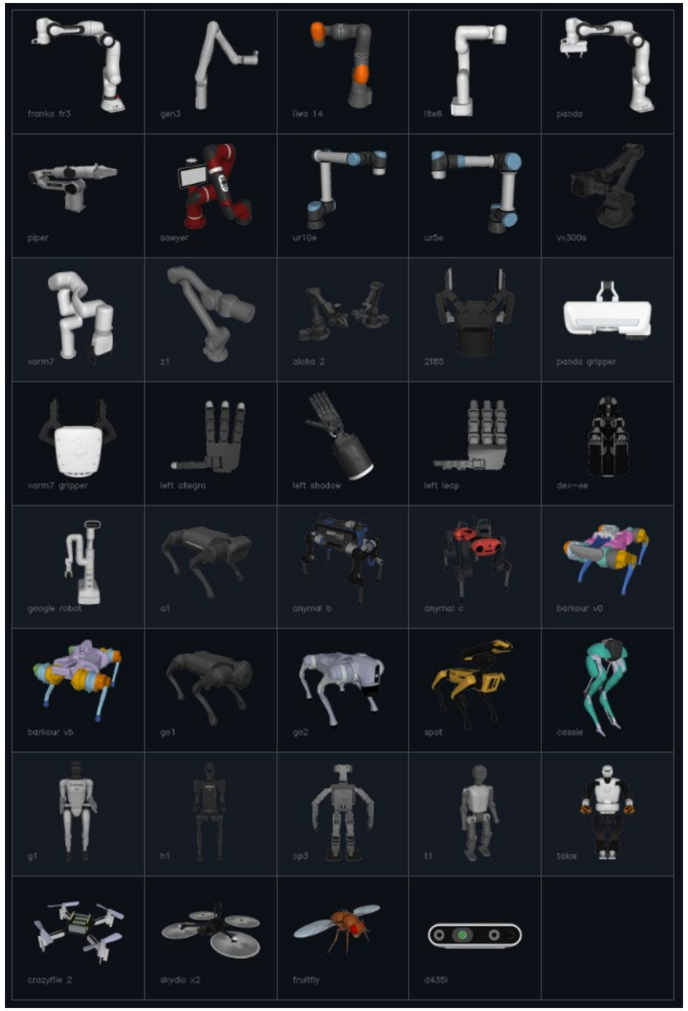
Google DeepMind

MuJoCo: A physics engine for model-based control

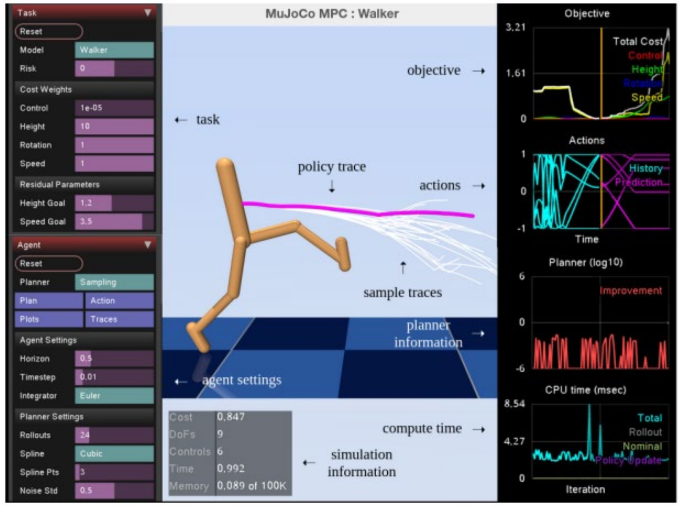
Emanuel Todorov, Tom Erez and Yuval Tassa
University of Washington

What is MuJoCo?

MuJoCo MENAGERIE

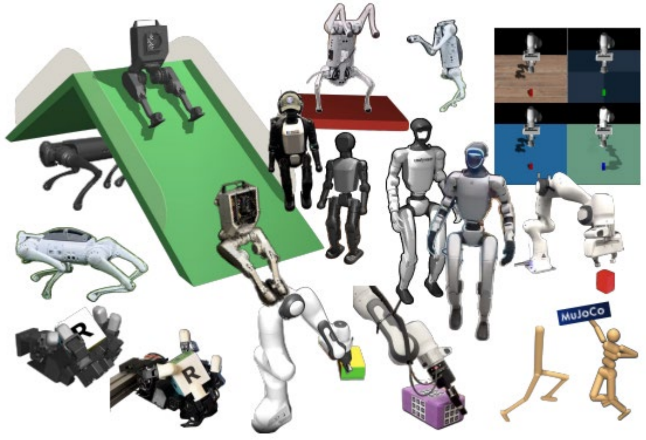


MuJoCo MPC



- Fully open-source interactive application & software framework for real-time predictive control
- Paper
- GitHub

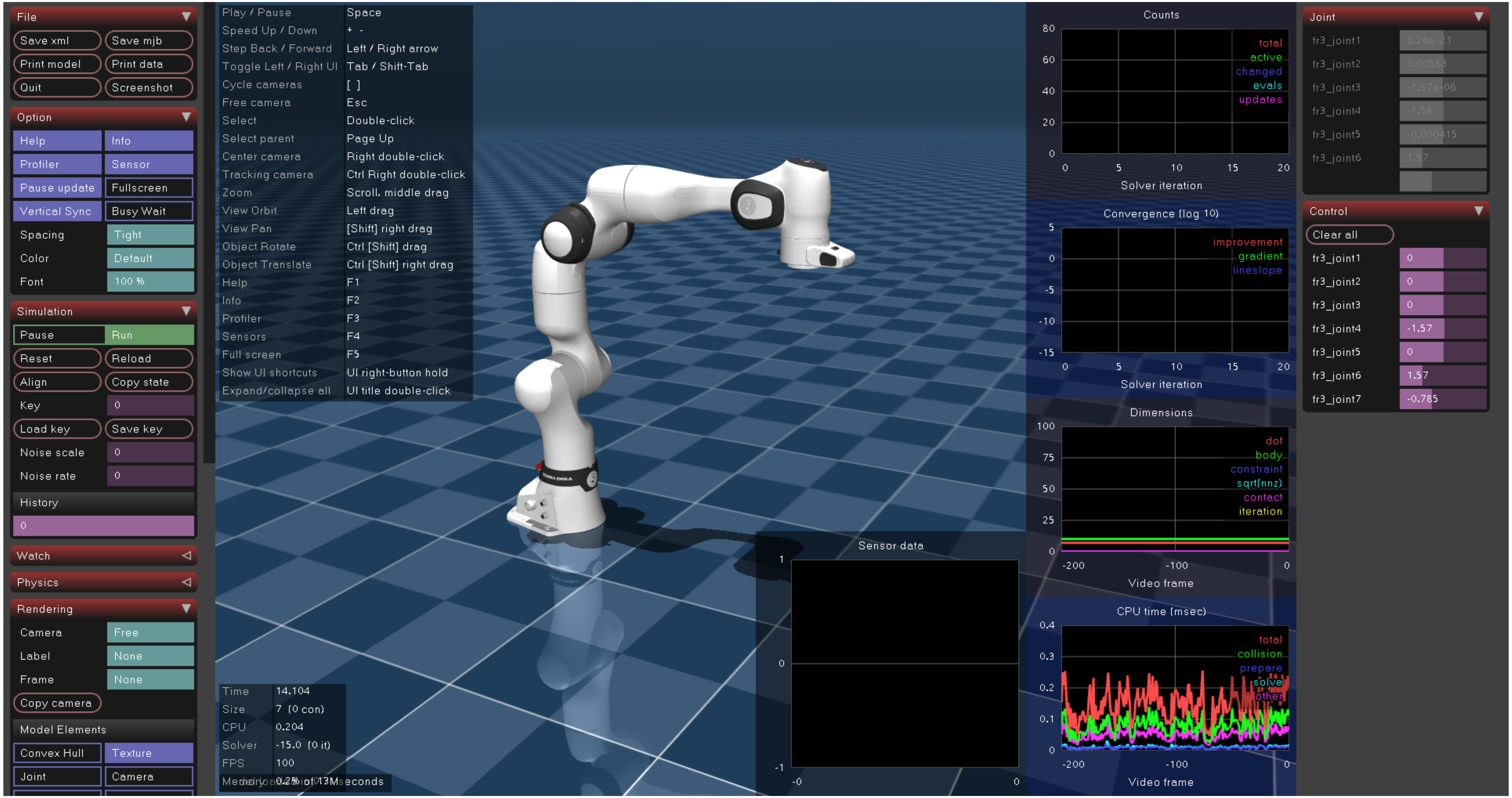
MuJoCo PLAYGROUND



- Fully open-source framework for robot learning build with MJX, with goal of simulation, training, and sim2real transfer
- Paper
- GitHub

Interactive GUI (Graphical User Interface)

MCL
Motion Control Lab



Robot Model Description: MJCF File

	MJCF (MuJoCo XML File)	*URDF (Unified Robot Description Format)*
Format	XML	
Usage	Simulation, Dynamics	Simulation, Dynamics, **ROS Standard**
Expressiveness	Supports **contacts, tendons, soft bodies, constraints**	Basic rigid-body structures
Joint & Contact	Supports **advanced dynamics: friction, damping**	Only basic joint definitions (no contact dynamics)
Joint Types	Supports **high degree-of-freedom for joints**: free, ball, slide, hinge	Basic: floating, prismatic, floating, planar
Modeling Flexibility	Supports more **flexible modeling** structure	Strict tree structure
Visualization	**Rich visual features**: materials, lights, camera	Basic mesh, color, texture
Conversion	MJCF → URDF (X)	URDF → MJCF (O)
Simulation Settings	Supports selection for timestep, integrator, solver options, etc.	-

Robot Model Description: MJCF File

MCL
Motion Control Lab

XML Reference

Introduction

This chapter is the reference manual for the MJCF modeling language used in MuJoCo.

XML schema

The table below summarizes the XML elements and their attributes in MJCF. Note that all information in MJCF is entered through elements and attributes. Text content in elements is not used; if present, the parser ignores it.

▼ **Collapse schema table**

The symbols in the second column of the table have the following meaning:

!	required element, can appear only once
?	optional element, can appear only once
*	optional element, can appear many times
R	optional element, can appear many times recursively

mujoco	!	model			
mujoco ↳ option	*	timestep ls_tolerance wind o_margin integrator iterations sdf_iterations	apirate noslip_tolerance magnetic o_solref cone ls_iterations sdf_initpoints	impratio ccd_tolerance density o_solimp jacobian noslip_iterations actuatorgroupdisable	tolerance gravity viscosity o_friction solver ccd_iterations
option ↳ flag	?	constraint contact warmstart sensor override multiccd	equality passive filterparent midphase energy island	frictionloss gravity actuation eulerdamp fwdinv nativeccd	limit clampctrl refsafe autoreset invdiscrete

MuJoCo XML Reference

Franka Robotics FR3 Description (MJCF)

Important

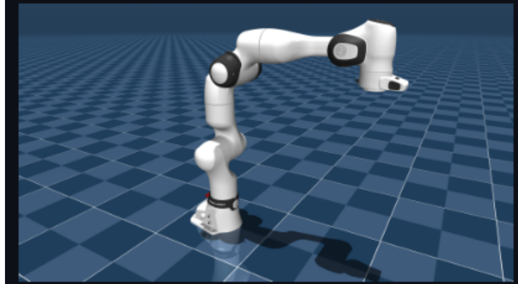
Requires MuJoCo 3.1.3 or later.

Changelog

See CHANGELOG.md for a full history of changes.

Overview

This package contains a simplified robot description (MJCF) of the Franka Research 3 (aka FR3) developed by Franka Robotics (formerly Franka Emika). It is derived from the publicly available URDF description.



Franka Research 3 (FR3) Model @ MuJoCo Menagerie

- Well-documented XML Reference
- Some model files for commercial robots have dependency on specific MuJoCo version

Programming API

API Reference

This chapter is the reference manual for the MuJoCo API. It is automatically kept in sync with MuJoCo's header files, but also contains additional information not available in the headers. The API is composed of 3 categories:

- Types
  - Primitive types
    - mjtNum
    - mjtByte
  - Enum types
    - Model
    - Data
    - Visualization
    - Rendering
    - User Interface
    - Spec
    - Plugins
  - Struct types
    - mjModel
    - mjOption
    - mjData
    - Auxiliary
    - Sim statistics
    - Visualisation
    - Rendering
    - User Interface
    - Model Editing
- Functions
  - Parse and compile
  - Main simulation
  - Support
  - Components
  - Sub components
  - Ray casting
  - Printing
  - Virtual file system
  - Initialization
  - Error and memory
  - Miscellaneous
  - Interaction
  - Visualization
  - OpenGL rendering
  - UI framework
  - Derivatives
  - Plugins
  - Threads
  - Standard math
  - Vector math
  - Sparse math
- Globals
  - Error callbacks
  - Memory callbacks
  - Physics callbacks
  - Collision table
  - String constants
  - Numeric constants
  - Macros
  - X Macros

MuJoCo API Reference

- Programming Language: C++ / Python
- Supports well-documented API
- **Built-in libraries for dynamics and math**
  - No need for external libraries like Pinocchio or RBDL

RBDL

Pinocchio
Efficient and versatile rigid body dynamics algorithms

Development Environment Set Up

Prerequisites for Programming Practice

- OS: **Ubuntu 20.04** (or higher distribution version would be fine, **Highly recommended**)
  - You can use **WSL2 in Windows**, but it is *not recommended for simulation rendering*
  - Example in this lecture is not supported for macOS users

- Programming Language: C++

- Build Tools: CMake, git, gcc, …

- Dependencies: Eigen (for Linear Algebra)

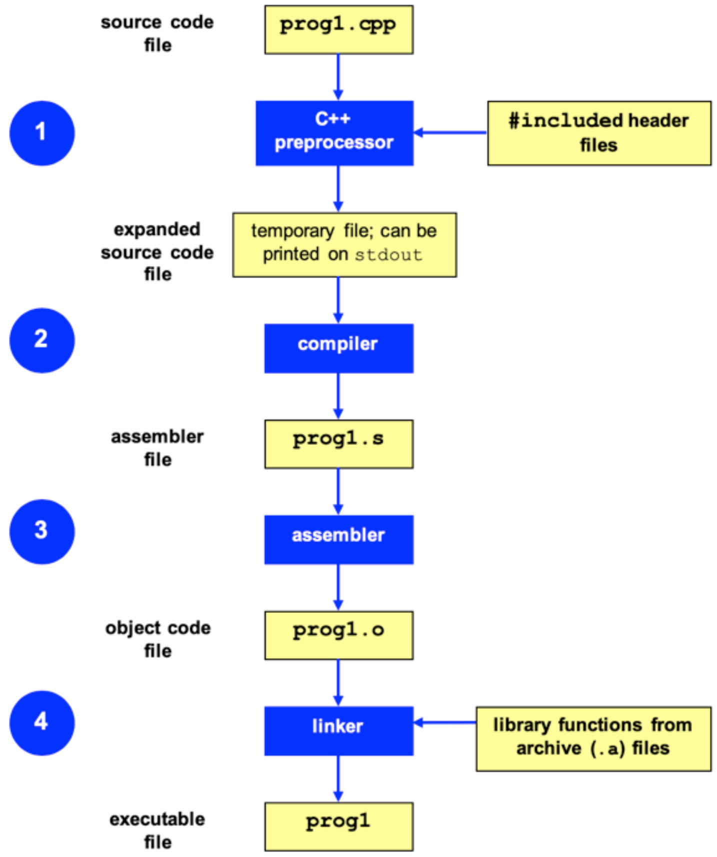
- Editor like Visual Studio Code or Cursor AI is recommended

Development Environment Set Up

Recommended Extensions

- Better Comments
- C/C++, C/C++ Extension Pack
- CMake, CMake Tools
- Clang-Format
- Git Graph
- Markdown All in One
- Material Icon Theme
- Rainbow Brackets
- XML Tools

Quick Review of C++ Build Process




```
gcc main.cpp -o my_program -lstdc++ -std=c++17
```

Compile using gcc in command line

CMake (Cross Platform Make)

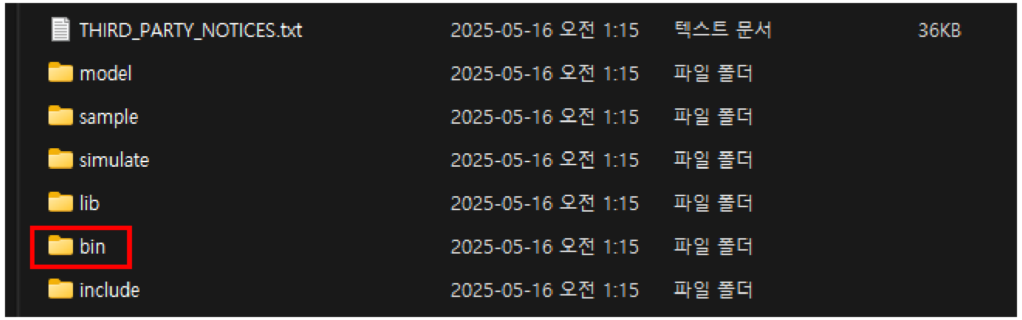
- Cross platform
- Support various compiler
- Dependency
- Modularization

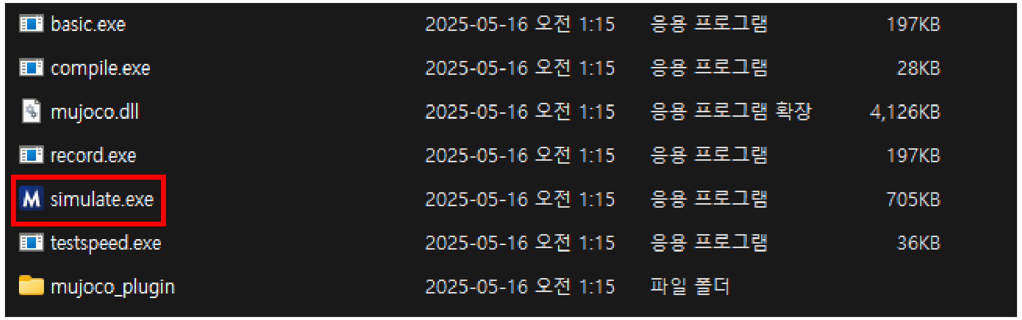
```
# Set minimum required version of cmake, project name and compile options
cmake_minimum_required(VERSION 3.16)
project(mujoco_cpp_template)

# set C/C++ Standard
if(NOT CMAKE_C_STANDARD)
  set(CMAKE_C_STANDARD 11)
endif()

if(NOT CMAKE_CXX_STANDARD)
  set(CMAKE_CXX_STANDARD 17)
endif()

# set repo's absolute directory
add_definitions(-DPROJECT_ROOT_DIR="${CMAKE_SOURCE_DIR}")

# set MuJoCo PATH defined in ~/.bashrc
set(MUJOCO_DIR $ENV{MUJOCO_PATH})
link_directories(${MUJOCO_DIR}/lib)

# find required packages
find_package(Eigen3 REQUIRED)

# set compile options
if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  # add_compile_options(-Wall -Wextra -Wpedantic) # when you can't find build error, try this
  add_compile_options(-Wpedantic)
endif()

# Find source and header files to create executable simulation file
file(GLOB_RECURSE SRCS
  ${CMAKE_CURRENT_SOURCE_DIR}/src/*.cc
  ${CMAKE_CURRENT_SOURCE_DIR}/src/*.cpp
)

# Create executable and link libraries
add_executable(${PROJECT_NAME} ${SRCS})

target_include_directories(${PROJECT_NAME}
PUBLIC
  ${CMAKE_CURRENT_SOURCE_DIR}/include
  ${MUJOCO_DIR}/include
  $<INSTALL_INTERFACE:include>
)

target_link_libraries(${PROJECT_NAME}
PRIVATE
  mujoco
  glfw
  pthread
)
```

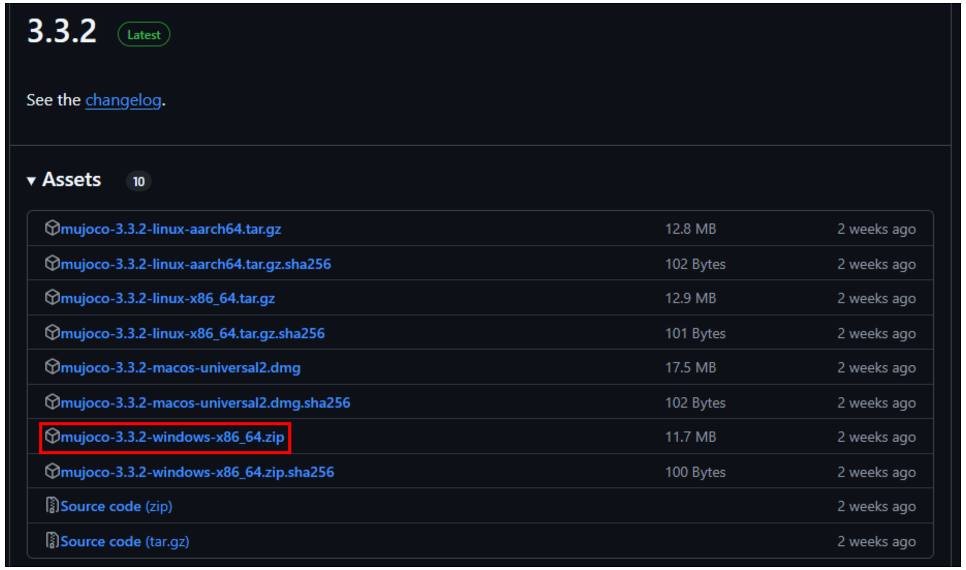
CMakeLists.txt

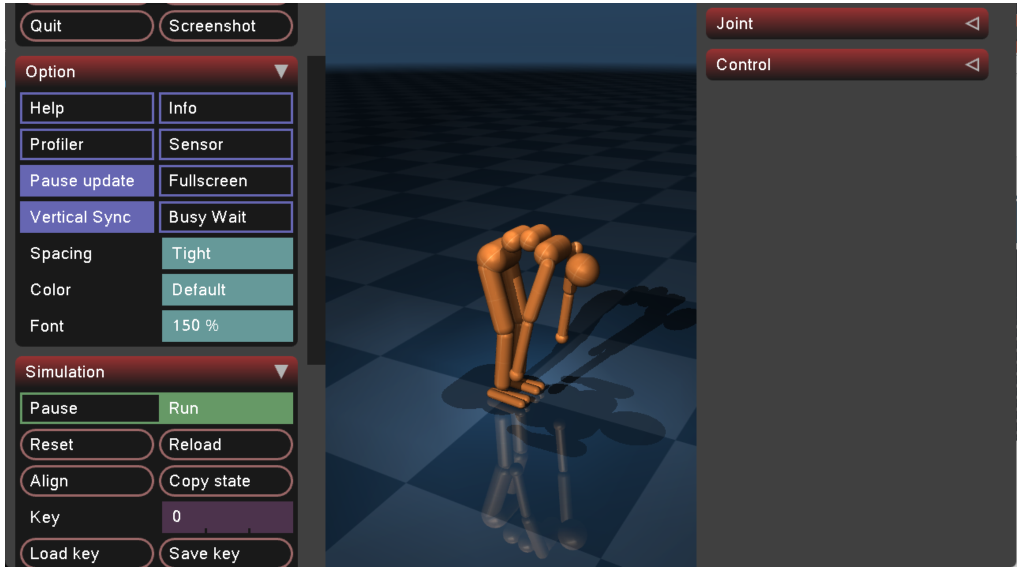
Installation (for Windows Users)

- Just to get feeling of MuJoCo in Windows…
  1. Go to [MuJoCo Releases]
  2. Download zip file for Windows
  3. Unzip file
  4. Run simulate.exe in `bin` folder
  5. Drag & Drop XML model file









Installation (for Linux & WSL2 Users)

- Lecture Material: https://github.com/jwhong1209/mujoco_cpp_template.git

Two Options for Getting Started

1. Set up and install all the requirements in your local PC

build-essentials

```
$ sudo apt update && sudo apt upgrade -y

$ sudo apt install build-essential
$ gcc --version # check if gcc is installed correctly

$ sudo apt-get install git
$ git --version

$ sudo apt install cmake
$ cmake --version
```

MuJoCo

```
# Clone and build MuJoCo from source
$ cd ~ # change path to your home directory
$ git clone https://github.com/deepmind/mujoco.git
$ cd ~/mujoco
$ mkdir -p install build
$ cd build
$ cmake ..
$ cmake .. -DCMAKE_INSTALL_PREFIX=~/mujoco/install
$ cmake --build .
$ cmake --install .

# Add environment variables to ~/.bashrc
$ echo 'export MUJOCO_PATH=$HOME/mujoco/install' >> ~/.bashrc
$ echo 'export LD_LIBRARY_PATH=$MUJOCO_PATH/lib:$LD_LIBRARY_PATH' >> ~/.bashrc
$ source ~/.bashrc
```

Eigen

```
$ sudo apt install libeigen3-dev # Then, eigen headers are installed at /usr/include/eigen3
```

Installation (for Linux & WSL2 Users)

- Lecture Material: https://github.com/jwhong1209/mujoco_cpp_template.git

Two Options for Getting Started

2. Use Docker

```
# Clone the repository
$ cd ~/<your-project-directory>
$ git clone https://github.com/jwhong1209/mujoco_cpp_template.git
$ cd mujoco_cpp_template

# Build and run Docker containter
$ docker build -t mujoco-cpp-template .
$ docker run -it --rm \
    -e DISPLAY=$DISPLAY \
    -v /tmp/.X11-unix:/tmp/.X11-unix \
    -v $(pwd):/workspace \
    mujoco-cpp-template
```

Practice Template

- Lecture Material: https://github.com/jwhong1209/mujoco_cpp_template.git

```
~/practice_ws/mujoco_cpp_template

|- assets
    |- data          →  Simulation data logging files: CSV, MATLAB
    |- model         →  MJCF files
|- common
    |- include       →  Common header / source files for robotics
    |- src
|- include           →  MuJoCo Simulation header / source files
|- src
|- .dockerignore
|- .gitignore
|- CMakeLists.txt
|- Dockerfile
|- README.md
```

How it works

- In **main.cc**, simulation is executed in background thread (physics_thread), while rendering in main thread (data is exchanged through mutex)

- In physics thread, both forward (FD) & inverse (ID) dynamics and then FD results (= acceleration) are integrated over specified timestep with chosen numerical integrator (e.g. Euler, RK4)
  - This is done by **`mj_step(m,d)`** function in physics loop
  - **`mj_forward(m,d)`** function internally call **`mjcb_control`** function in which **control law is implemented**

```
// simulate until t = 10 seconds
while( d->time<10 )
  mj_step(m, d);
```

```
void mj_step(const mjModel* m, mjData* d) {
  // common to all integrators
  mj_checkPos(m, d);
  mj_checkVel(m, d);
  mj_forward(m, d);
  mj_checkAcc(m, d);

  // compare forward and inverse solutions if enabled
  if( mjENABLED(mjENBL_FWDINV) )
    mj_compareFwdInv(m, d);

  // use selected integrator
  if( m->opt.integrator==mjINT_RK4 )
    mj_RungeKutta(m, d, 4);
  else
    mj_Euler(m, d);
}
```

```
// install control callback
mjcb_control = mycontroller;
```

Equation of Motion

$$M\dot{v} + c = \tau + J^T f$$

Forward Dynamics

$$\dot{v} = M^{-1}(\tau + J^T f - c)$$

Inverse Dynamics

$$\tau = M\dot{v} + c - J^T f$$

Practice

Wrap-ups

MCL
Motion Control Lab

- MuJoCo is a **lightweight yet powerful physics engine** designed for **high-fidelity simulation of robot dynamics and control**

- It provides:
  - **Intuitive interactive GUI** for visualization and debugging
  - **Rich XML-based modeling language (MJCF)** for defining robots and environments
  - **Built-in dynamics API** that eliminates the need for external libraries like RBDL or Pinocchio

- By enabling **fast, reproducible, and physically accurate simulations**, MuJoCo allows safe prototyping, testing, and training of robot controllers — essential for modern model-based and learning-based control

- In short, *MuJoCo helps bridge the gap between theory and practice in robot control*