BEST OVERVIEW

<https://mujoco.readthedocs.io/en/latest/programming.html>

DOCS

<https://mujoco.readthedocs.io/en/latest/overview.html>

This is the prebuilt binaries

To see example go here and download:

<https://github.com/deepmind/mujoco/releases>

Then go to bin and:

./simulate ../model/humanoid/humanoid.xml

Programming

Mujoco py

<https://github.com/openai/mujoco-py>

<https://github.com/openai/mujoco-py#ubuntu-installtion-troubleshooting>

<https://mujoco.readthedocs.io/en/latest/programming.html>

<https://mujoco.readthedocs.io/en/latest/programming.html#saderivative>

Derivative cc gives example of forward and inverse dynamics

Finally, calling mj\_forward which corresponds to the abstract dynamics function f(t,x,u) computes the time-derivative of the state vector. The corresponding fields of mjData are

ACTUALLY WRITING CONTROL LOOP

<https://mujoco.readthedocs.io/en/latest/programming.html#simulation-loop>

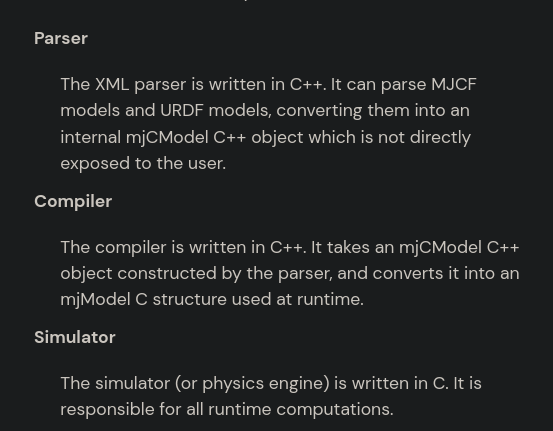
Next we turn to the controls and applied forces. The control vector in MuJoCo is

u = (mjData.ctrl, mjData.qfrc\_applied, mjData.xfrc\_applied)

These quantities specify control signals (mjData.ctrl) for the actuators defined in the model, or directly apply forces and torques specified in joint space (mjData.qfrc\_applied) or in Cartesian space (mjData.xfrc\_applied).

We wont have any free ball joints but hey good to know

In the presence of quaternions (i.e. when free or ball joints are used), the position vector mjData.qpos has higher dimensionality than the velocity vector mjData.qvel and so this is not a simple time-derivative in the sense of scalars, but instead takes quaternion algebra into account.



Reinforcement Learning with MuJoCo

MuJoCo is a physics engine for detailed, efficient rigid body simulations with contacts. It has a dynamic library with C/C++ API. mujoco-py allows using MuJoCo from Python 3.

It includes an XML parser, model compiler, simulator, and interactive OpenGL visualizer.

Open AI

Open ai gym builds on top of mujoco-py?

<https://gym.openai.com/docs/>

<https://gym.openai.com/envs/#classic_control>

<https://gym.openai.com/envs/#mujoco>

The registry:

gym’s main purpose is to provide a large collection of environments that expose a common interface and are versioned to allow for comparisons. To list the environments available in your installation, just ask gym.envs.registry

Simulation params

The computation of constraint forces and constrained accelerations involves solving an optimization problem numerically. MuJoCo has three algorithms for solving this optimization problem: CG, Newton, PGS. Each of them can be applied to a pyramidal or elliptic model of the friction cones, and with dense or sparse constraint Jacobians.

<https://mujoco.readthedocs.io/en/latest/modeling.html#algorithms-and-related-settings>

Also gives recs on which friction model to use for what types of models

Modeling

<https://mujoco.readthedocs.io/en/latest/modeling.html>

<https://mujoco.readthedocs.io/en/latest/modeling.html#tips-and-tricks>

<https://mujoco.readthedocs.io/en/latest/XMLreference.html>

As explained in [Model instances](https://mujoco.readthedocs.io/en/latest/overview.html#instance) in the Overview chapter, MuJoCo models can be loaded from plain-text XML files in the MJCF or URDF formats, and then compiled into a low-level mjModel.

Even though loading and compilation are presently combined in one step, compilation is independent of loading, meaning that the compiler works in the same way regardless of how mjCModel was created.

This makes it possible to design models interactively, by re-loading often and visualizing the changes. Note that the [simulate.cc](https://github.com/deepmind/mujoco/blob/main/sample/simulate.cc) code sample has a keyboard shortcut for re-loading the current model (Ctrl+L).

Can also save back to xml with c api call [mj\_saveLastXML](https://mujoco.readthedocs.io/en/latest/APIreference.html#mj-savelastxml)

An MJCF model can consist of multiple (included) XML files as well as meshes, height fields and textures referenced from the XML. After compilation, the contents of all these files are assembled into mjModel, which can be saved into a binary MJB file with [mj\_saveModel](https://mujoco.readthedocs.io/en/latest/APIreference.html#mj-savemodel). The MJB is a stand-alone file and does not refer to any other files. It also loads faster. So we recommend saving commonly used models as MJB and loading them when needed for simulation.

Actuators

Automatic computation of actuator length ranges is done at compile time, and the results are stored in mjModel.actuator\_lengthrange of the compiled model.

These are *not* separate model elements. Internally MuJoCo supports only one actuator type

As explained in the [Actuation model](https://mujoco.readthedocs.io/en/latest/computation.html#geactuation) section of the Computation chapter, MuJoCo offers a flexible actuator model with transmission, activation dynamics and force generation components that can be specified independently. The full functionality can be accessed via the XML element [general](https://mujoco.readthedocs.io/en/latest/XMLreference.html#general) which allows the user to create a variety of custom actuators. In addition, MJCF provides shortcuts for configuring common actuators. This is done via the XML elements [motor](https://mujoco.readthedocs.io/en/latest/XMLreference.html#motor), [position](https://mujoco.readthedocs.io/en/latest/XMLreference.html#position), [velocity](https://mujoco.readthedocs.io/en/latest/XMLreference.html#velocity), [cylinder](https://mujoco.readthedocs.io/en/latest/XMLreference.html#cylinder), [muscle](https://mujoco.readthedocs.io/en/latest/XMLreference.html#muscle). T

UR5 Modeling

<https://github.com/wangcongrobot/dual_ur5_husky_mujoco>

Ok the other mujoco project does not have any arm joints

Ok this is conversion steps:

<https://github.com/wangcongrobot/dual_ur5_husky_mujoco#step3-verify-urdf-file>

<https://github.com/wangcongrobot/dual_ur5_husky_mujoco#step4-convert-urdf-file-to-mjcf-file>

Original arm urdf

<https://github.com/ros-industrial/universal_robot/tree/kinetic-devel/ur_description/urdf>

I remember this this will do the subs for the xacro to give me the final urdf

For melodic

rosrun xacro xacro --inorder -o model.urdf model.urdf.xacro

For noetic

rosrun xacro xacro ur5.urdf.xacro > ur5.urdf

Forgot about this

a.urdf.xacro only defines the hunter xacro macro, but that macro is never called (and thus your robot never instantiated).

I typically create a 'top level' .xacro file for this (something like hunter.xacro fi) which is empty, apart from a call to the macro. In your case:

<?xml version="1.0" ?>

<robot name="hunter" xmlns:xacro="http://ros.org/wiki/xacro">

<xacro:include filename="$(find hunter\_pkg)/a.urdf.xacro"/>

<xacro:hunter/>

</robot>

Muscle actuators

Just for my own stuff

<https://mujoco.readthedocs.io/en/latest/modeling.html#muscle-actuators>

Relation to OpenSim

The standard software used by researchers in biomechanics is OpenSim. We have designed our muscle model to be similar to the OpenSim model where possible, while making simplifications which result in significantly faster and more stable simulations. To help MuJoCo users convert OpenSim models, here we summarize the similarities and differences.

The activation dynamics model is identical to OpenSim, including the default time constants.

The FLV function is not exactly the same, but both MuJoCo and OpenSim approximate the same experimental data, so they are very close. For a description of the OpenSim model and summary of relevant experimental data, see:

Millard et al, “Flexing computational muscle: modeling and simulation of musculotendon dynamics”, J Biomech Eng. 2013 Feb;135(2)

We assume inelastic tendons while OpenSim can model tendon elasticity. We decided not to do that here, because tendon elasticity requires fast-equilibrium assumptions which in turn require various tweaks and are prone to simulation instability.