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. Recap of Last Lecture

2. Building 2-DoF Pendulum Model
   - Why 2-DoF Pendulum Model?
   - Key Features of MJCF File

3. Controller Implementation
   - Controller Structure Overview
   - Model Update: Kinematics / Dynamics
   - Trajectory Generation
   - Control Law: Computed Torque Control (CTC)

4. Beyond Basics: Customization and Integration
   - Building Your Own Robot Model
   - MuJoCo UI Customization and ROS Integration

5. Wrap-ups

Recap of Last Lecture

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

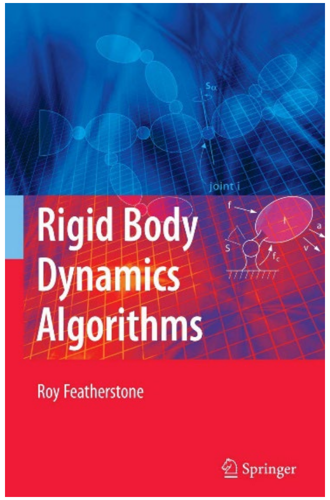
- It provides:
  - **Intuitive interactive GUI** for visualization and debugging
  - **Rich XML-based modeling language (MJCF)** for defining robots and environments
  - **Built-in dynamics API** that eliminates the need for external libraries like RBDL or Pinocchio

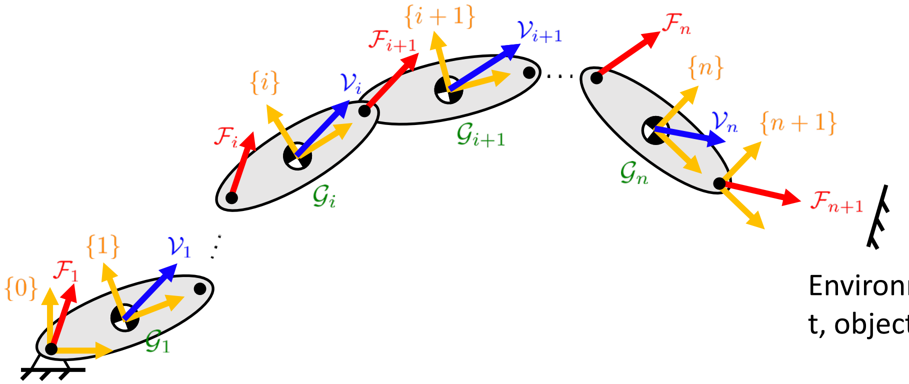
- By enabling **fast, reproducible, and physically accurate simulations**, MuJoCo allows safe prototyping, testing, and training of robot controllers — essential for modern model-based and learning-based control

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

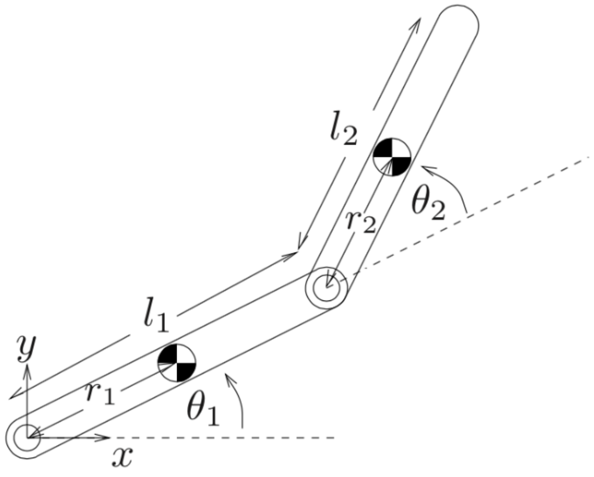
Why 2-DoF Pendulum Model?

To analyze Robot Dynamics … $M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \longrightarrow \hat{M}(q)\ddot{q} + \hat{C}(q,\dot{q})\dot{q} + \hat{g}(q) = \tau$





Environmen
t, object



- Rare system that you can **describe and validate dynamics by hand** (e.g. inverse kinematics)
- Good template for practicing robot dynamics and control

RBDL

Pinocchio
Efficient and versatile rigid body dynamics algorithms

Key Features of MJCF File

MCL
Motion Control Lab

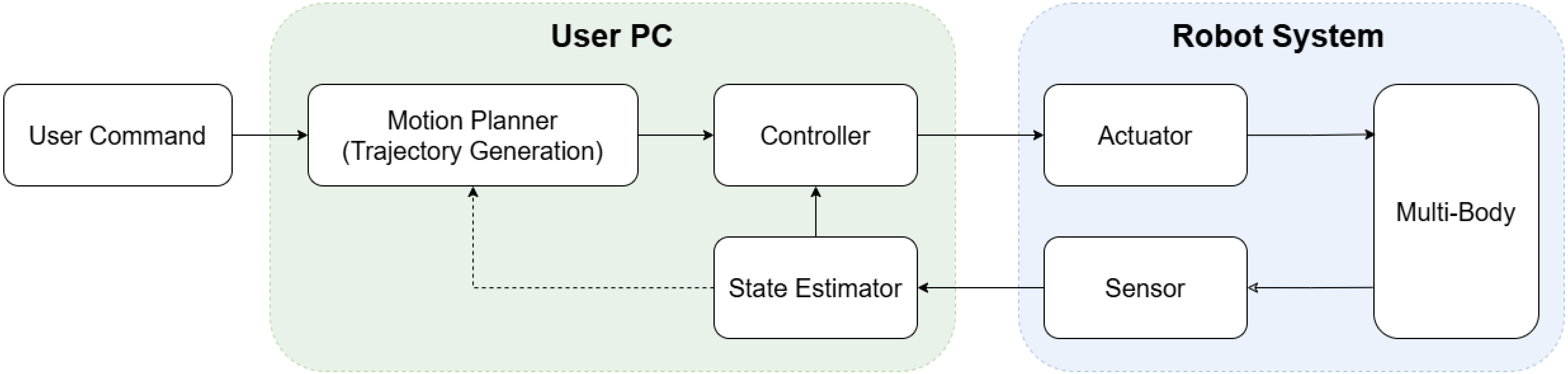
double_pendulum.xml

```xml
<mujoco model="double_pendulum">
  <compiler angle="radian" />

  <option integrator="implicitfast" timestep="0.002"/>

  <asset>
    <material name="black" rgba="0.0 0.0 0.0 1.0" />
    <material name="blue" rgba="0.0 0.0 1.0 1.0" />
    <material name="red" rgba="1.0 0.0 0.0 1.0" />
  </asset>

  <worldbody>
    <body name="base_link" pos="0 0 0">
      <body name="link1" pos="0 0.5 1.5" euler="-1.5708 0 0">
        <inertial pos="0 0 0" euler="0 0 0" mass="1.0" diaginertia="0.03 0.04 0.05"/>
        <geom type="box" size="0.05 0.05 0.5" material="blue"/>
        <joint name="joint1" type="hinge" pos="0 0 -0.5" axis="1 0 0"/>

        <body name="link2" pos="0.1 0 1.0">
          <inertial pos="0 0 0" euler="0 0 0" mass="1.0" diaginertia="0.07 0.06 0.08"/>
          <geom type="box" size="0.05 0.05 0.5" material="red"/>
          <joint name="joint2" type="hinge" pos="0 0 -0.5" axis="1 0 0"/>
          <site name ="ee_site" type="sphere" pos="0 0 0.5" size="0.1" rgba="0 0.9 0 0.5" />
        </body>
      </body>
    </body>
  </worldbody>

  <actuator>
    <motor joint="joint1" name="motor1" gear="1"/>
    <motor joint="joint2" name="motor2" gear="1"/>
  </actuator>

  <keyframe>
    <key name="home" qpos="-0.7854 1.5708" ctrl="0 0"/>
  </keyframe>
</mujoco>
```

scene.xml

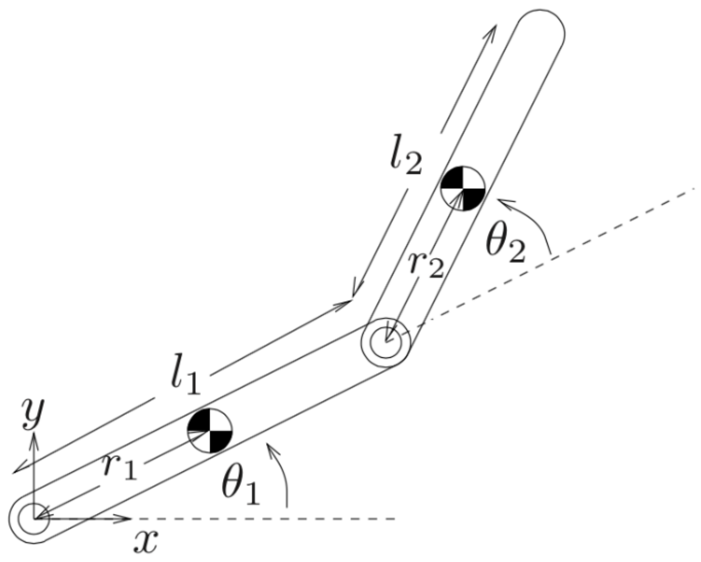
```xml
<mujoco model="dbpen scene">
  <include file="double_pendulum.xml"/>

  <statistic center="0.2 0 0.4" extent=".8"/>

  <visual>
    <headlight diffuse="0.6 0.6 0.6" ambient="0.3 0.3 0.3" specular="0 0 0"/>
    <rgba haze="0.15 0.25 0.35 1"/>
    <global azimuth="120" elevation="-20"/>
  </visual>

  <asset>
    <texture type="skybox" builtin="gradient" rgb1="0.3 0.5 0.7" rgb2="0 0 0" width="512" height="3072"/>
    <texture type="2d" name="groundplane" builtin="checker" mark="edge" rgb1="0.2 0.3 0.4" rgb2="0.1 0.2 0.3"
      markrgb="0.8 0.8 0.8" width="300" height="300"/>
    <material name="groundplane" texture="groundplane" texuniform="true" texrepeat="5 5" reflectance="0.2"/>
    <material name="wall_material" rgba="0.55 0.27 0.07 1.0"/>
  </asset>

  <worldbody>
    <light pos="0 0 1.5" dir="0 0 -1" directional="true"/>
    <geom name="floor" size="0 0 0.05" type="plane" material="groundplane"/>
    <geom name="wall" pos="0 3.2 2.0" size="2.0 0.5 2.0" type="box" material="wall_material"/>
  </worldbody>

  <sensor>
    <jointpos name="joint1_pos" joint="joint1" noise="50e-9"/> <!-- 0 -->
    <jointpos name="joint2_pos" joint="joint2" noise="50e-9"/> <!-- 1 -->

    <jointvel name="joint1_vel" joint="joint1" noise="50e-9"/> <!-- 2 -->
    <jointvel name="joint2_vel" joint="joint2" noise="50e-9"/> <!-- 3 -->

    <framepos    name="ee_pos"    objtype="site" objname="ee_site" /> <!-- 4, 5, 6 -->
    <framequat   name="ee_quat"   objtype="site" objname="ee_site" /> <!-- 7, 8, 9, 10 -->
    <framelinvel name="ee_linvel" objtype="site" objname="ee_site" /> <!-- 11, 12, 13 -->
    <frameangvel name="ee_angvel" objtype="site" objname="ee_site" /> <!-- 14, 15, 16 -->
    <framelinacc name="ee_linacc" objtype="site" objname="ee_site" /> <!-- 17, 18, 19 -->
    <frameangacc name="ee_angacc" objtype="site" objname="ee_site" /> <!-- 20, 21, 22 -->

    <force  site="ee_site" name="ee_force"  noise="0" /> <!-- 23, 24, 25 -->
    <torque site="ee_site" name="ee_torque" noise="0" /> <!-- 26, 27, 28 -->
  </sensor>
</mujoco>
```

Controller Structure Overview



- **Various components (= modules)** are necessary for robot system control
  → *Object-oriented Programming (OOP)*

Model Update: Kinematics / Dynamics



- **Kinematics**
  - End-effector Position

$$p_x = l_1 \cos(q_1) + l_2 \cos(q_{12})$$
$$p_y = l_1 \sin(q_1) + l_2 \sin(q_{12}) \qquad q_{12} = q_1 + q_2$$

  - End-effector Velocity

$$\begin{bmatrix} v_x \\ v_y \end{bmatrix} = \boxed{\begin{bmatrix} -l_1 \sin(q_1) - l_2 \sin(q_{12}) & -l_2 \sin(q_{12}) \\ l_1 \cos(q_1) + l_2 \cos(q_{12}) & l_2 \cos(q_{12}) \end{bmatrix}} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

$$= J(q) \quad : \text{Jacobian matrix}$$

- **Statics** $\tau = J^T F$

- **Dynamics**

$$\hat{M}(q) = \begin{bmatrix} I_1 + m_1 d_1^2 + I_2 + m_2 d_2^2 + m_2 l_1^2 + 2 m_2 l_1 d_2 \cos(q_2) & I_2 + m_2 d_2^2 + m_2 l_1 d_2 \cos(q_2) \\ I_2 + m_2 d_2^2 + m_2 l_1 d_2 \cos(q_2) & I_2 + m_2 d_2^2 \end{bmatrix}$$

$$= \begin{bmatrix} J_1 + J_2 + m_2 l_1^2 + 2 m_2 l_1 d_2 \cos(q_2) & J_2 + m_2 l_1 d_2 \cos(q_2) \\ J_2 + m_2 l_1 d_2 \cos(q_2) & J_2 \end{bmatrix}$$

$$\hat{C}(q, \dot{q})\dot{q} = \begin{bmatrix} -m_2 l_1 d_2 \sin(q_2) \cdot \dot{q}_2 & -m_2 l_1 d_2 \sin(q_2) \cdot (\dot{q}_1 + \dot{q}_2) \\ m_2 l_1 d_2 \sin(q_2) \cdot \dot{q}_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} = \begin{bmatrix} -m_2 l_1 d_2 \sin(q_2) \cdot (q_2^2 + 2 q_1 q_2) \\ m_2 l_1 d_2 \sin(q_2) \cdot q_1^2 \end{bmatrix}$$

$$\hat{g}(q) = \begin{bmatrix} g(m_1 d_1 + m_2 l_1) \cos(q_1) + g m_2 d_2 \cos(q_{12}) \\ g m_2 d_2 \cos(q_{12}) \end{bmatrix}$$

Cubic Polynomial

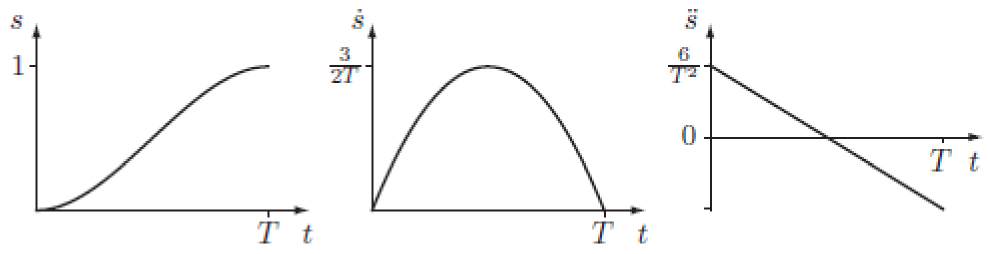
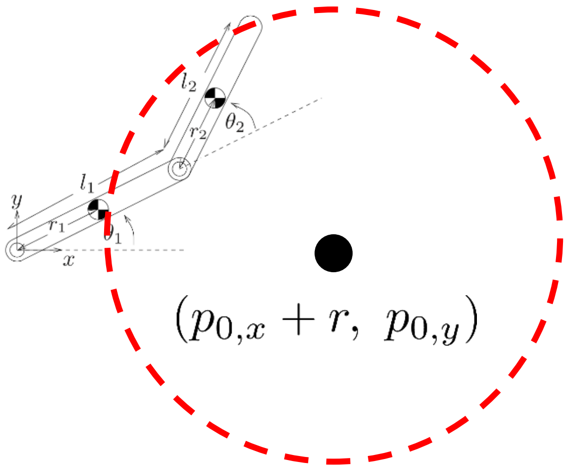


Figure 9.3: Plots of $s(t)$, $\dot{s}(t)$, and $\ddot{s}(t)$ for a third-order polynomial time scaling.

$$s(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2 + a_3(t - t_0)^3$$

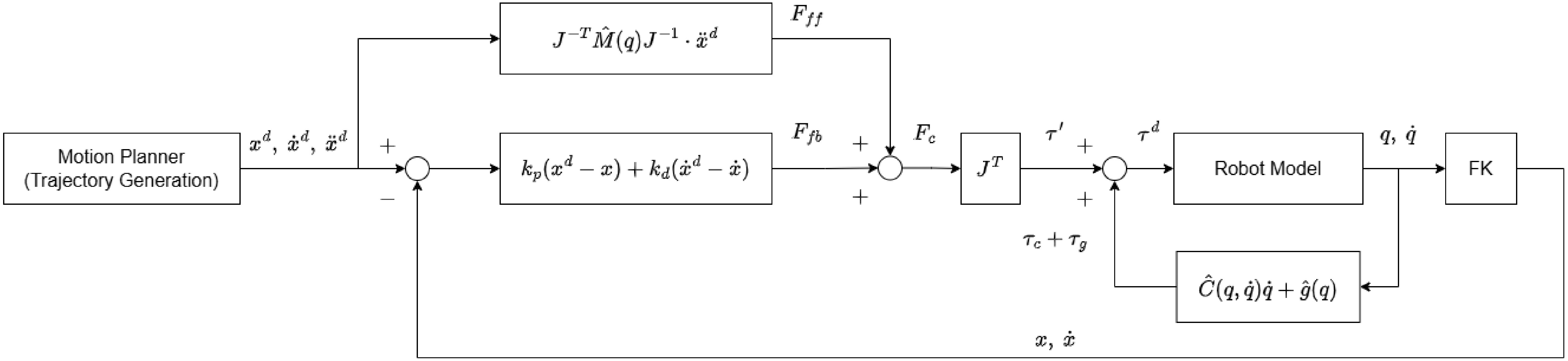
$$\begin{aligned} s(t_0) &= p_0 \\ \dot{s}(t_0) &= v_0 \\ s(t_f) &= p_f \\ \dot{s}(t_f) &= v_f \end{aligned} \longrightarrow \begin{aligned} a_0 &= p_0 \\ a_1 &= v_0 \\ a_2 &= \frac{3(p_f - p_0)}{(t_f - t_0)^2} - \frac{2v_0 + v_f}{t_f - t_0} \\ a_3 &= \frac{-2(p_f - p_0)}{(t_f - t_0)^3} + \frac{v_0 + v_f}{(t_f - t_0)^2} \end{aligned}$$

Cartesian-space Circular Motion



$$\begin{aligned} p_x &= p_{0,x} + r \cdot (1 - \cos(\omega(t - t_0))) \\ p_y &= p_{0,y} + r \cdot \sin(\omega(t - t_0)) \end{aligned}$$

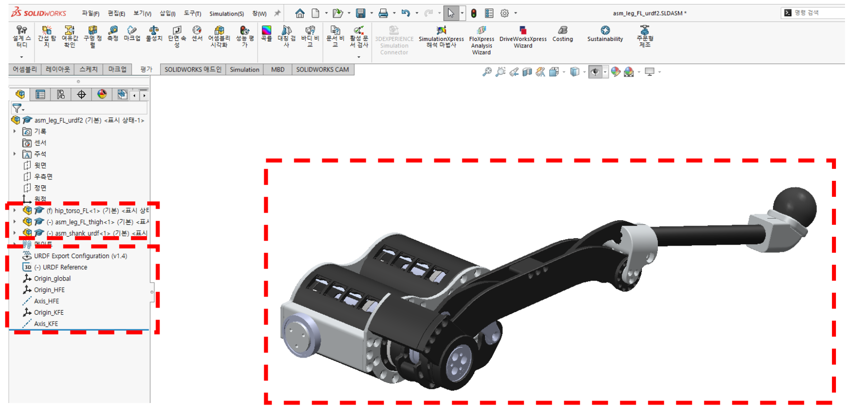
Computed Torque Control (CTC) in Cartesian-space



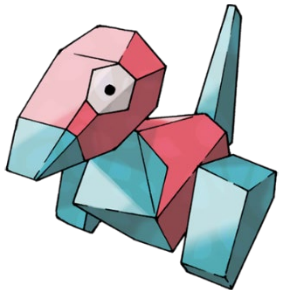
- Kinematics / Dynamics values with *hat* represent numerical (or estimated) values

Building Your Own Robot Model

1. Import URDF & Mesh (STL) File from CAD Data



2. Convert URDF → MJCF or Write by yourself
   - It is highly recommended to **separate body** into **visual part** and **collision part**



quadMCL_legFL.xml

```xml
<mujoco model="quadMCL_legFL">
  <compiler angle="radian" meshdir="urdf_legFL/meshes/" />

  <option timestep="0.0001" integrator="RK4" >
    <flag sensornoise="enable" energy="enable" contact="enable" />
  </option>

  <option gravity = "0 0 -9.81" cone="elliptic" impratio="100" />

  <asset>
    <material name="dark" specular="0" shininess="0.25" rgba="0.2 0.2 0.2 1" />
      <mesh name="hip_torso_FL" file="hip_link.STL" />
      <mesh name="thigh_FL" file="thigh_link.STL" />
      <mesh name="shank_FL" file="shank_link.STL" />
  </asset>

  <worldbody>
    <light diffuse=".5 .5 .5" pos="0 0 3" dir="0 0 -1"/>
      <body name="hipTorso_FL" pos="0 0 0.1" euler="-1.57 0 0">
        <inertial pos="0.10195 0.026085 0.00046109" euler="0 0 0" mass="1.047" diaginertia="0.00068326 0.00082943 0.00087739"/>
        <geom type="mesh" contype="1" conaffinity="0" group="1" material="dark" mesh="hip_torso_FL" />
        <joint name="torso" type="slide" pos="0 0 0" axis="0 -1 0" />

        <site type="box" name="hip_torso" pos="0.10195 0.026085 0.0004610" size="0.1 0.08 0.05" rgba="1 1 1 0.0" />

        <body name="thigh_FL" pos="0 0 0" euler="0 0 0">
          <inertial pos="0.00966 0 0.108119" euler="0 0 0" mass="0.50316"
            diaginertia="839220.12E-10 11869737.27E-10 12268062.32E-10" />
          <geom type="mesh" contype="1" conaffinity="0" group="1" material="dark" mesh="thigh_FL" />
          <joint name="HFE_FL" type="hinge" pos="0.1435 0 0.07105" axis="0 0 -1" />

          <body name="shank_FL" pos="0 0 0">
            <inertial pos="-0.24034 0 0.108119" euler="0 0 0" mass="0.28735" diaginertia="413501.87E-10 7681064.07E-10 7682017.13E-10
            <geom type="mesh" contype="1" conaffinity="0" group="2" material="dark" mesh="shank_FL" />
            <joint name="KFE_FL" type="hinge" pos="-0.1065 0 0.07105" axis="0 0 -1" />

            <site type="sphere" name="foot" pos="-0.3565 0 0.108119" size="0.05" rgba="1 1 1 0.0" />
          </body>
        </body>
      </body>
  </worldbody>

  <actuator>
    <motor joint="HFE_FL" name="torque_HFE" gear="1" ctrllimited="true" ctrlrange="-100 100" />
    <position name="pos_servo_HFE" joint="HFE_FL" kp="100" />
    <velocity name="vel_servo_HFE" joint="HFE_FL" kv="10" />

    <motor joint="KFE_FL" name="torque_KFE" gear="1" ctrllimited="true" ctrlrange="-100 100" />
    <position name="pos_servo_KFE" joint="KFE_FL" kp="100" />
    <velocity name="vel_servo_KFE" joint="KFE_FL" kv="10" />
  </actuator>

  <sensor>
    <framelinvel objtype="site" objname="foot" />
    <touch name="touch_sensor" site="foot" />
    <force name="grf_sensor" site="foot" />

    <framepos objtype="site" objname="hip_torso" />
    <framelinvel objtype="site" objname="hip_torso" />
  </sensor>
</mujoco>
```

MuJoCo UI Customization & ROS Integration

MuJoCo Interactive UI

- Camera view
- Perturbation
- …

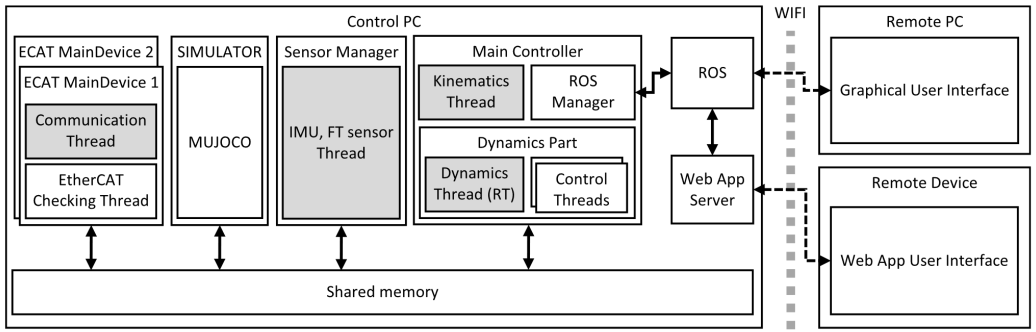
main.cc

```
//* UI settings *//
sim->ui0_enable = false;         // left UI is disabled (TAB)
sim->ui1_enable = false;         // right UI <is disabled (Shift + TAB)
sim->pending_.load_key = true;  // load key frame
// sim->run = false;
```

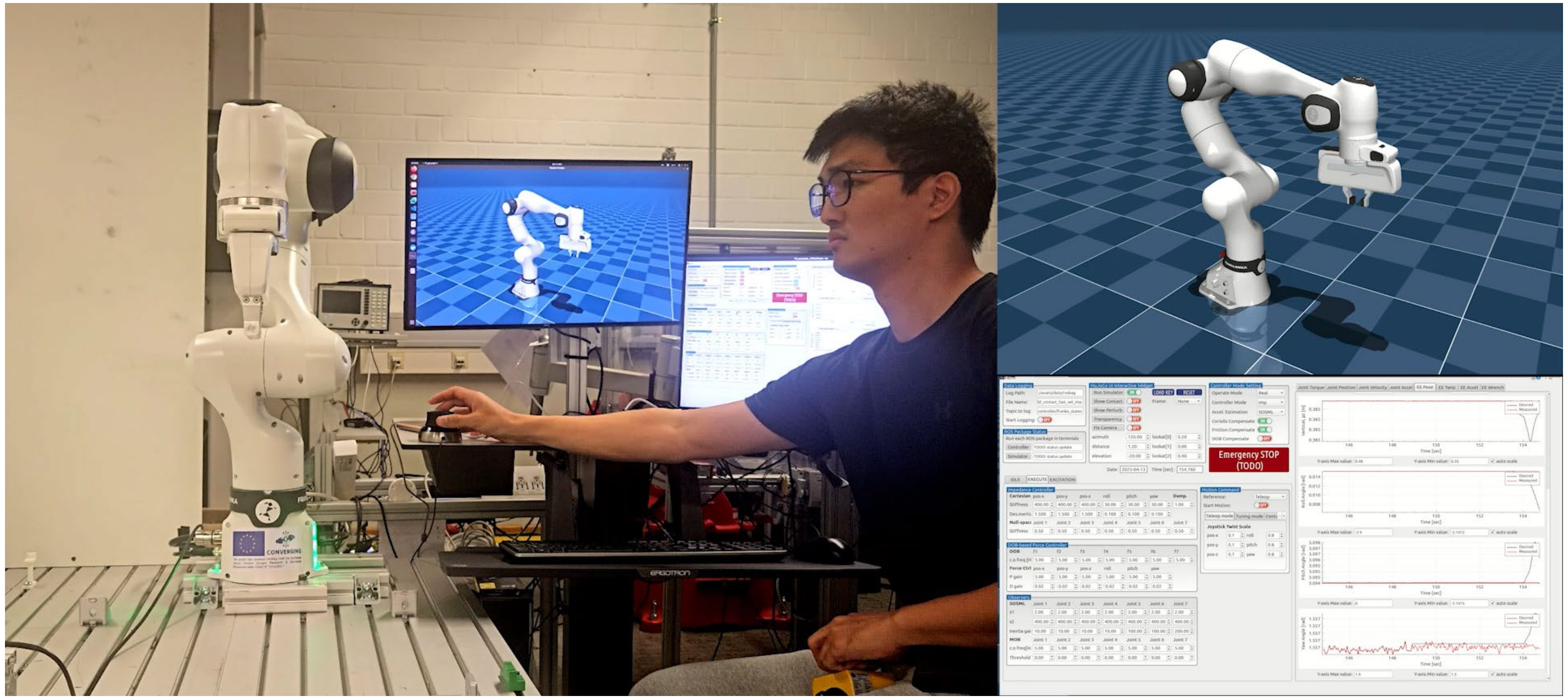
simulate.cc

```
//* ------------------------------------------------------------------------------------------
//* ----- VISUALIZATION ------------------------------------------------------------------------
/* Camera View Settings */
// printf("Current Camera View: %f, %f, %f, %f, %f, %f \n", this->cam.azimuth,
//  this->cam.distance, this->cam.elevation, this->cam.lookat[0], this->cam.lookat[1],
//  this->cam.lookat[2]);
this->cam.azimuth = 180;
this->cam.distance = 5;
this->cam.elevation = -4.75;
this->cam.lookat[0] = 0.2;
this->cam.lookat[1] = 0.8;
this->cam.lookat[2] = 0.5;

/* Perturbation */
this->opt.flags[mjVIS_PERTFORCE] = true;
//* ------------------------------------------------------------------------------------------
```

ROS Integration



J. Ahn et al, "Dual-Channel EtherCAT Control System for 33-DOF Humanoid Robot TOCABI," IEEE Access 2023



Wrap-ups

- 2-DoF pendulum model is good template for practicing robotics and control

- Generally, implement (1) state estimator (kinematics/dynamics), (2) motion planner, and (3) controller

- You can customize your simulation GUI using API

- You can build your own robot in MuJoCo
  - Separate your model into Visual part & Collision part