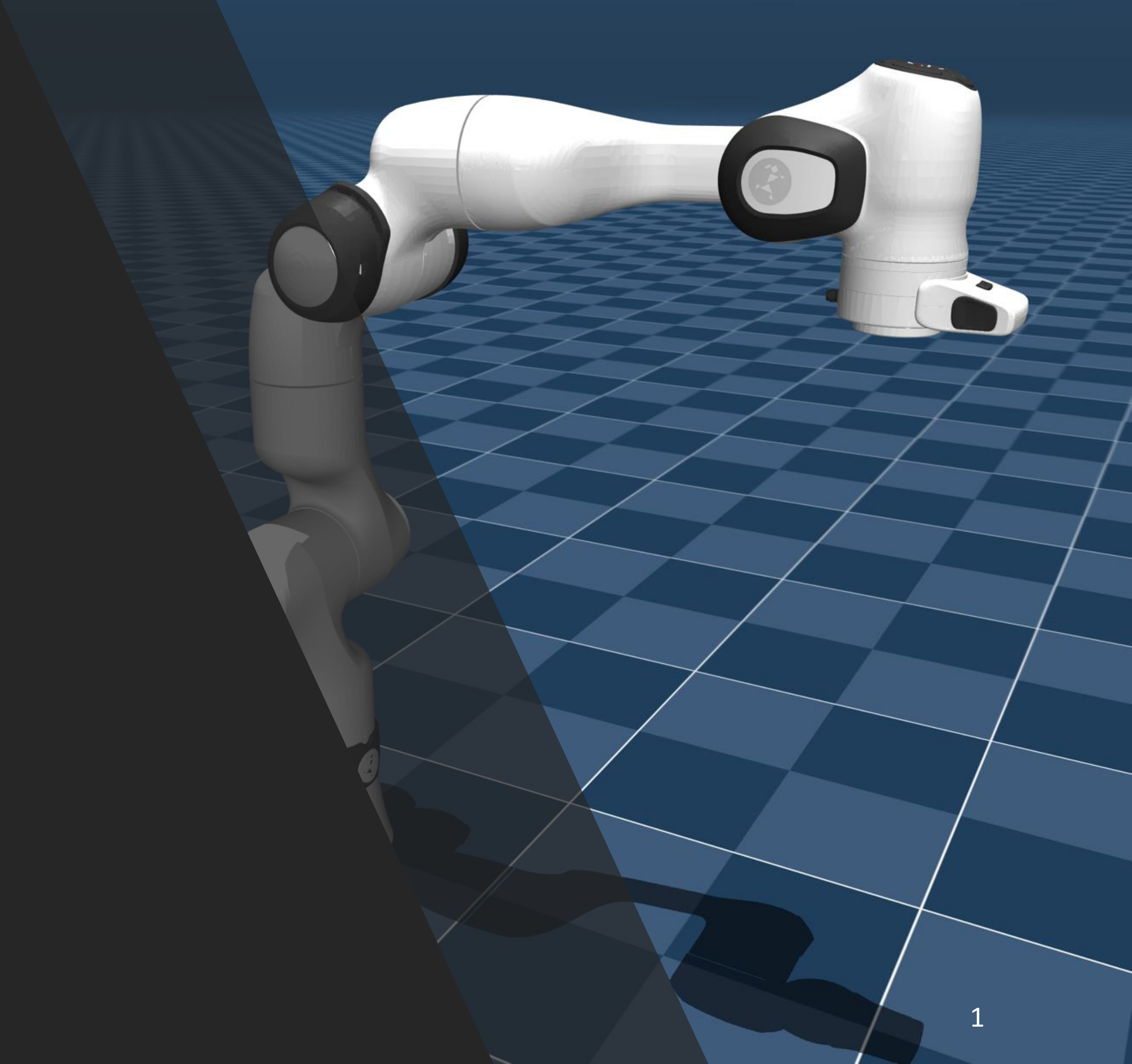
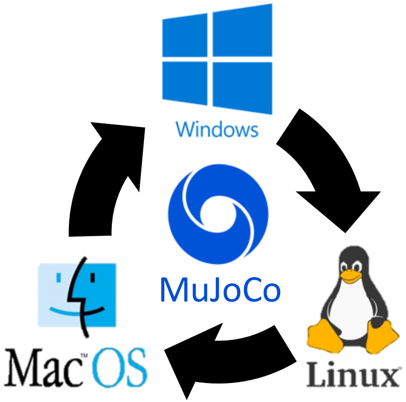
Convergent Robotics Technology

MuJoCo Tutorial



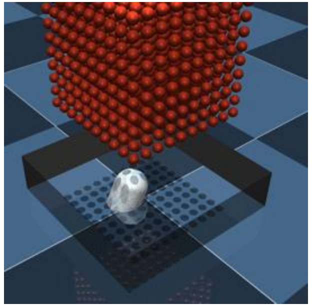
1.1 Introduction to MuJoCo



① OS usage

② API functions

③ Support various languages



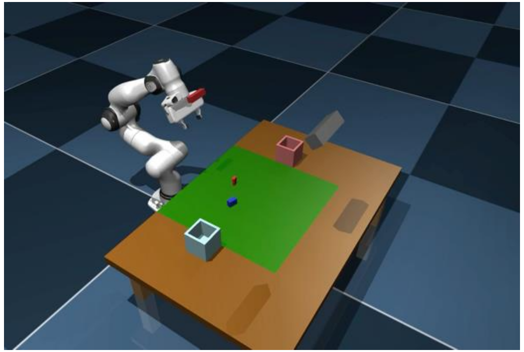
④ Advanced physics engine

⑤ Model customization

⑥ Extensive utilities

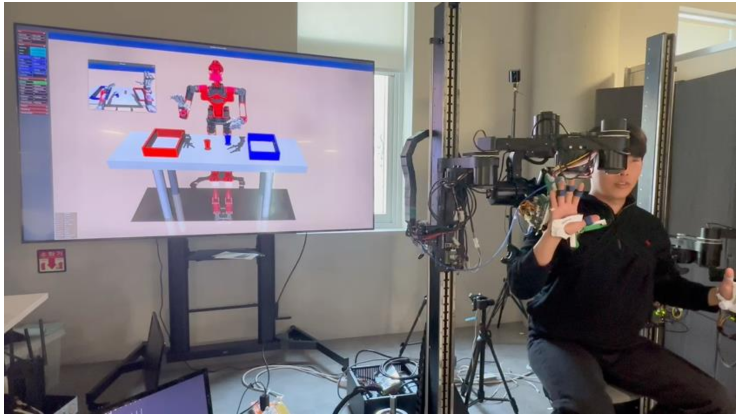
DYROS

1.1 Introduction to MuJoCo



Pick and Place with Manipulator

RL based Walking Humanoid



Teleoperation Humanoid by Haptic device

Example video for running MuJoCo

1.1 Introduction to MuJoCo

- Why MuJoCo?
  - Erez et.al., “Erez, Tom, Yuval Tassa, and Emanuel Todorov. "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx”, IEEE ICRA, 2015
    - Performance evaluation between MuJoCo and the other Physics Engine (ex. ODE, PhysX) in the perspective as
      - ✓ Raw Timing
        - How fast the engines update simulation steps per second in CPU.
      - ✓ Consistency
        - How much the simulation deviates from a reference trajectory as timestep increases.
      - ✓ Speed-Accuracy Trade-off
        - How simulation accuracy changes as computation speed increases.
      - ✓ Energy & Momentum Conservation
        - How well the engine preserves energy and momentum in a frictionless system.
      - ✓ Grasp Stability
        - How large a timestep can be used while maintaining a stable grasp.

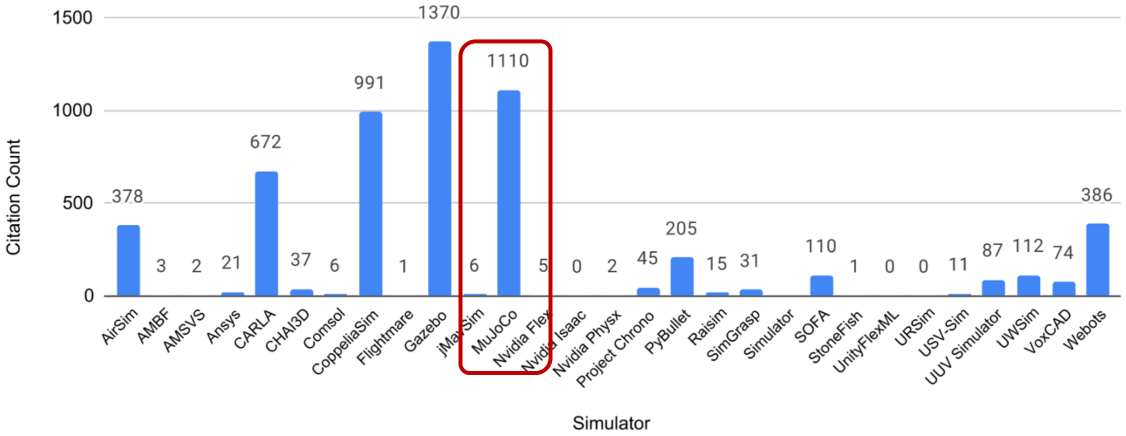
1.1 Introduction to MuJoCo

- **Why MuJoCo?**
  - **Erez et.al., "Erez, Tom, Yuval Tassa, and Emanuel Todorov. "Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx", IEEE ICRA, 2015**

	MuJoCo	Bullet	ODE	Havok	PhysX
Raw Timing	**Fastest** in robotics models	**Slowest**	**Fast** in object-heavy scenes	**Moderate speed**	**Moderate speed**
Consistency	**Best**, minimal deviation	**Unstable** at large timesteps	**Unstable** in grasping	**Low accuracy**	**Lowest accuracy**
Speed Accuracy Trade-off	**Best balance**, highly efficient	**Low accuracy** at high speed	**Accurate but slow**	**Fast but inaccurate**	**Very fast but least accurate**
Conservation	**Best** in energy and momentum	**Poor** conservation	**Good** for rotation but not energy	**Weak conservation**	**Poor conservation**
Grasp Stability	Holds object at **large timesteps** (16ms)	**Unstable** (1/32ms)	**Unstable** (1/4ms)	Not tested	**Partially stable** (2ms)

1.1 Introduction to MuJoCo

- **Why MuJoCo?**
  - **J. Collins et. al., “A Review of Physics Simulators for Robotic Applications ”, IEEE Access, 2021**
    - **Citation count from 2016 to 2020 for reviewed simulators.**

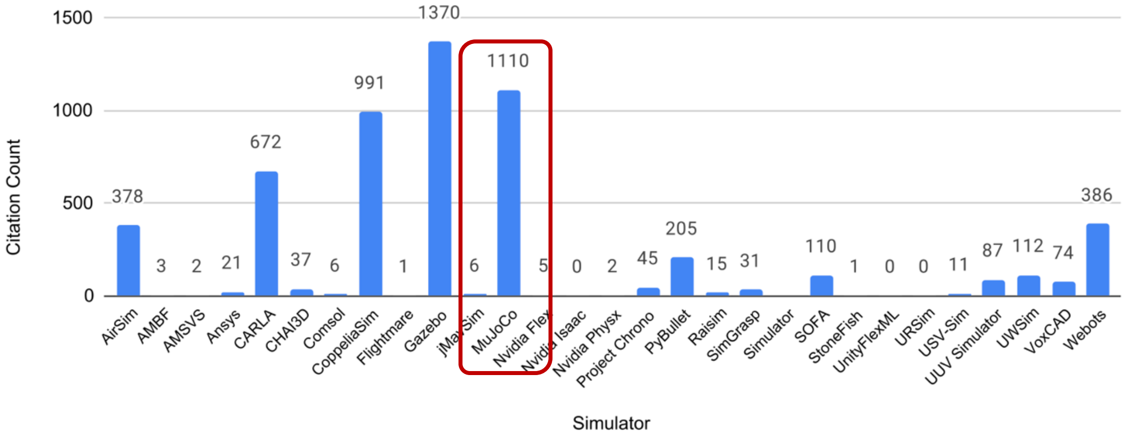


✓ **MuJoCo is one of the most actively used simulators in the field of robotics.**

DYROS

1.1 Introduction to MuJoCo

- **Why MuJoCo?**
  - **J. Collins et. al., “A Review of Physics Simulators for Robotic Applications ”, IEEE Access, 2021**
    - **Citation count from 2016 to 2020 for reviewed simulators.**



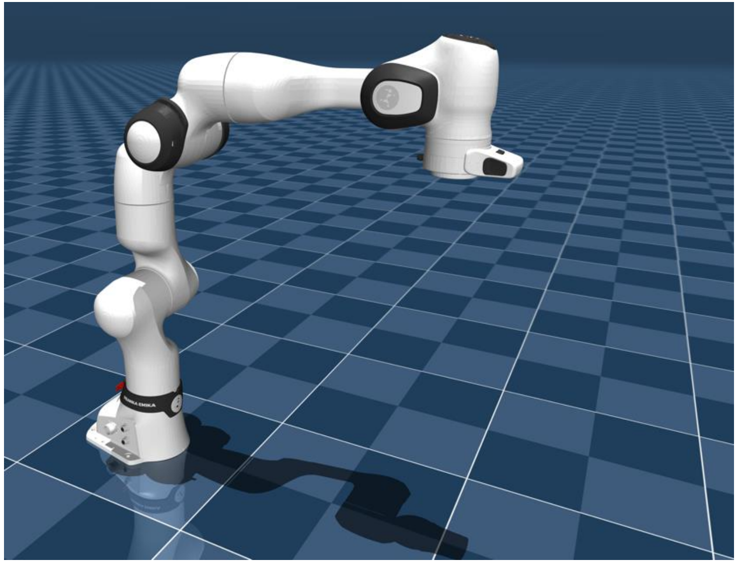
✓ **The biggest advantage of using MuJoCo, as reported, is its superior contact stability compared to other simulators.**

DYROS

1.2 MuJoCo Modeling (MJCF)

```
<mujoco model="fr3">
  <compiler angle="radian" meshdir="assets"/>
  <option integrator="implicitfast"/>

  <default>
    <default class="fr3">
      <joint armature="0.1" damping="1"/>
      …
  </default>

  <asset>
    <material name="black" rgba=".2 .2 .2 1"/>
    …
  </asset>

  <worldbody>
    <body name="base" childclass="fr3">
      <body name="fr3_link0">
        …
  </worldbody>

  <actuator>
    <position class="fr3" name="fr3_joint1"
joint="fr3_joint1" kp="4500" kv="450"/>
    …
  </actuator>

</mujoco>
```

fr3.xml



FR3 on MuJoCo

MuJoCo FR3 model code: https://github.com/google-deepmind/mujoco_menagerie/tree/main/franka_fr3

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

DYROS

1.2 MuJoCo Modeling (MJCF)

```xml
<mujoco model="fr3">
  <compiler angle="radian" meshdir="assets"/>
  <option integrator="implicitfast"/>

  <default>
    <default class="fr3">
      <joint armature="0.1" damping="1"/>
      …
  </default>

  <asset>
    <material name="black" rgba=".2 .2 .2 1"/>
     …
  </asset>

  <worldbody>
    <body name="base" childclass="fr3">
      <body name="fr3_link0">
        …
  </worldbody>

  <actuator>
    <position class="fr3" name="fr3_joint1"
joint="fr3_joint1" kp="4500" kv="450"/>
    …
  </actuator>

</mujoco>
```

fr3.xml

Main XML elements to define model:

- mujoco
- compiler
- option
- compiler
- asset
- (world)body
- actuator
- …

MuJoCo FR3 model code: https://github.com/google-deepmind/mujoco_menagerie/tree/main/franka_fr3

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

DYROS

1.2 MuJoCo Modeling (MJCF)

```
<mujoco model="fr3">
  <compiler angle="radian" meshdir="assets"/>
  <option integrator="implicitfast"/>
    …
</mujoco>
```

- mujoco
  : Unique top-level element for identifying XML file as MJCF
  - model: string
    : Name of the model

- compiler
  : Options for the built-in parser and compiler
  - angle: [radian, degree]
    : Unit of angles for expressing MJCF model
  - meshdir: string
    : File path to mesh and height field files

- option
  : Options for simulation
  - integrator: [Euler, RK4, implicit, implicitfast]
    : Numerical integrator to be used

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

DYROS

1.2 MuJoCo Modeling (MJCF)

```
<asset>
    <material name="black" rgba=".2 .2 .2 1"/>
     …
    <mesh file="link0_0.obj"/>
     …
  </asset>
```

- **asset**

  : Grouping elements for defining assets

  - **material**

    : Material asset

    - **name: string**

      : Name of the material

    - **rgba: real(4)**

      : Color and transparency of the material

  - **mesh**

    : Mesh asset

    - **file: string**

      : Path where mesh file is

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

DYROS

1.2 MuJoCo Modeling (MJCF)

```
<worldbody>
    <body name="base" childclass="fr3">
    …
      <body name="fr3_link1">
         <inertial pos "…" quat="…" mass"…"
diaginertia="…"/>
         <joint name="fr3_joint1" axis="0 0 1"
range="…" actuatorfrcrange=”…"/>
         <geom name="fr3_link1_collision"
class="collision" mesh="link1_coll"/>
        …
  </worldbody>
```

- **(world)body**
  : Model body to construct kinematic tree
  - **name: string**
    : Name of the body
  - **inertial**
    : Mass and inertial properties of the body
    - **pos: real(3)**
      : Position of the inertia frame
    - **quat: real(4)**
      : Orientation of the inertia frame
    - **mass: real**
      : Mass of the body
    - **diaginertia: real(3)**
      : Diagonal inertia matrix
    - **file: string**
      : Path where mesh file is

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

DYROS

1.2 MuJoCo Modeling (MJCF)

```
<worldbody>
    <body name="base" childclass="fr3">
    …
      <body name="fr3_link1">
         <inertial pos "…" quat="…" mass"…"
diaginertia="…"/>
         <joint name="fr3_joint1" axis="0 0 1"
range="…" actuatorfrcrange="…"/>
         <geom name="fr3_link1_collision"
class="collision" mesh="link1_coll"/>
        …
  </worldbody>
```

- **(world)body**

  : Model body to construct kinematic tree

  - **name: string**

    : Name of the body

  - **joint**

    : Joint between two body

    - **name: string**

      : Name of the joint

    - **axis: real(3)**

      : Axis of rotation for hinge joints / Direction of translation for slide joints

    - **range: real(2)**

      : Joint limit

    - **actuactorfrcrange: real(2)**

      : Clamping values for force limitation

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

DYROS

1.2 MuJoCo Modeling (MJCF)

```
<worldbody>
    <body name="base" childclass="fr3">
    …
      <body name="fr3_link1">
         <inertial pos "…" quat="…" mass"…"
diaginertia="…"/>
         <joint name="fr3_joint1" axis="0 0 1"
range="…" actuatorfrcrange=”…"/>
         <geom name="fr3_link1_collision"
class="collision" mesh="link1_coll"/>
        …
  </worldbody>
```

- **(world)body**
  : Model body to construct kinematic tree
  - **name: string**
    : Name of the body
  - **geom**
    : 3D shapes rigidly attached to the body
    - **name: string**
      : Name of the geom
    - **class: string**
      : Default class for setting unspecified attributes
    - **mesh: string**
      : Name of the mesh asset to be instantiated

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

1.2 MuJoCo Modeling (MJCF)

```
<actuator>
    <position class="fr3" name="fr3_joint1"
joint="fr3_joint1" kp="4500" kv="450"/>
    …
  </actuator>
```

- **actuator**

  : Grouping element for actuator definitions

  - **position**

    : Position servo actuator element

    - **class: string**

      : Active default class

    - **name: string**

      : Name of the actuator

    - **joint: string**

      : Determine the type of actuator transmission by specifying the joint it acts on.

    - **kp, kv: real**

      : Gain for position feedback and damping

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

DYROS

1.2 MuJoCo Modeling (MJCF)

```xml
<mujoco model="fr3_scene">
  <include file="fr3.xml"/>

  <visual>
    <headlight diffuse="0.6 0.6 0.6"
     …
</mujoco>
```

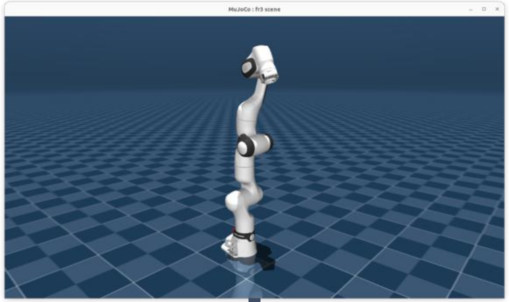
scene.xml

```python
import mujoco as mj
import mujoco.viewer

m = mj.MjModel.from_xml_path(scene.xml)
d = mj.MjData(m)

with mujoco.viewer.launch_passive(m, d, …) as viewer:
  while viewer.is_running() and not self.quit:
    mujoco.mj_step(m, d)
    …
```

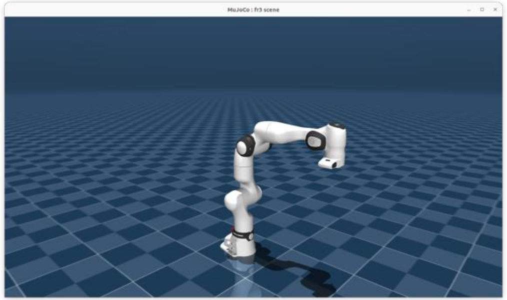
Python code

For more details, refer this site. https://mujoco.readthedocs.io/en/latest/modeling.html

MuJoCo API on Python & C++



Implement here

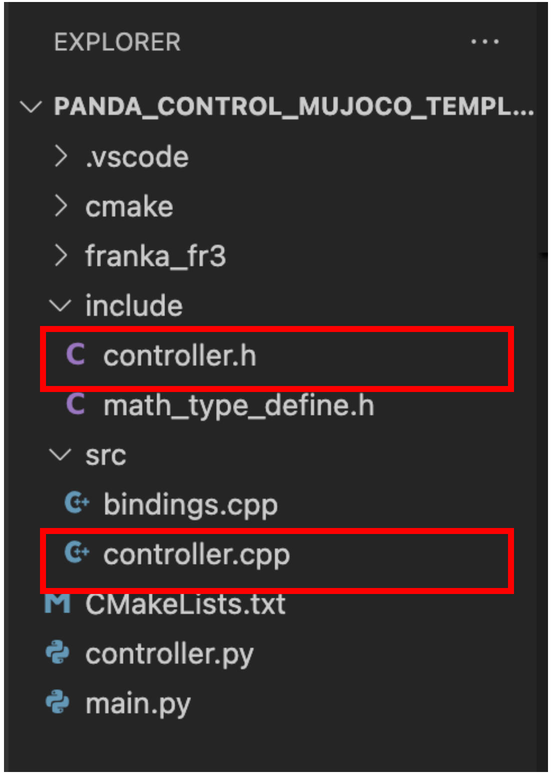


Read data → Compute algorithm → Write data → Update simulation

Python | **C++** | **Python**

DYROS

Project



- You can do your homework by modifying these files.

DYROS

Software architecture

main(python) / RobotController(python)

- Console input/output

cRoboticsController(cpp)

- Robot control algorithm
- Contains mathematical variables and algorithms

MuJoco

- MuJoCo interface
- Sends joint commands

Software architecture

cRoboticsController(cpp)

```
cRoboticsController::keyMapping(const int& key);

cRoboticsController::updateModel(const VectorXd& q, const VectorXd& q);
cRoboticsController::printState(const VectorXd& q, const VectorXd& q);
cRoboticsController::compute(const double& play_time);
cRoboticsController:: getCtrlInput();
```

→ **Implement Here**

DYROS

keyMapping()

- Controller.cpp

```
void cRoboticsController::keyMapping(const int &key)
{
    switch(key)
    {
      …
    // ########### implement here ##############
    case ‘5’:
        setMode(Controller_enum_name);
        break;
      …
    }
}
```

- Controller.h

```
enum CTRL_MODE{
…
// ########### implement here ##############
Controller_enum_name,
…
}
```

- Define user input from keyboard. (**add** setMode statements) `setMode(mode_name)`
- `mode_name` is a `enum` variable. (**add** name of controller)
- `control_mode_` of `cRoboticsController` is changed to `mode_name`

DYROS

updateModel()

- Controller.cpp

```
void cRoboticsController::updateModel(const VectorXd &q,
                                      const VectorXd &qdot,
                                      const VectorXd &tau)
{
    …
    {
      …
      if(!updateKinematics(q_, qdot_)) return false;
      if(!updateDynamics(q_, qdot_)) return false;
      …
    }
}
```

- Controller.h

```
// Current joint space state
VectorXd q_;
VectorXd qdot_;
VectorXd tau_;

// Current task space state
Matrix4d x_;
VectorXd xdot_;
MatrixXd J_;
```

```
// Current joint space
dynamics
MatrixXd M_;
VectorXd g_;
VectorXd c_;
```

- Getting joint states $(q, \dot{q}, \tau)$ from MuJoCo
- Update Model states w.r.t. the joint states by updateKinemtics and updateDynamics.
- You can utilize these terms for your controller.

DYROS

printState( )

- Controller.cpp

```
void cRoboticsController::printState()
{
 // ########## implement here #############
    std::cout << …
}
```

- Print user-defined log
- Frequency with 2 Hz

compute()

- Controller.cpp

```
void cRoboticsController::compute(const double& play_time)
{
    …
    switch(control_mode_)
    {
        // ########## implement here #############
        case(Controller_enum_name):
        {
            …
            // q_desired_ =
            // torque_desired_ =
            // logging_file_ << … << std::endl;

        }
    }
    …
}
```

- Define user input.
  (**add** switch statements with pre-defined `enum` data)
- Control input depends on control mode
  - Position → `q_desired_`
  - Torque → `torque_desired_`
- Logging data saved as .txt

DYROS

math_type_define.h

1. cubic
   - Generate trajectory using cubic spline
   - joint angles, position in the task(operational) space

2. rotationCubic
   - Generate trajectory using cubic spline
   - rotation matrix

3. getPhi
   - compute orientation error in task(operational) space

Questions

- Simulation & Robot exp. TA

  윤준헌 – yoonjh98@snu.ac.kr

Q&A