# Modeling and Simulation of Dynamic Systems using MuJoCo

**Prof. Jamie Paik**
**Dr. Yuhao Jiang**
**Reconfigurable Robotics Laboratory**
**EPFL, Switzerland**

# Topics

- Introduction to Dynamic Modeling
  - What is dynamic modeling
  - Why we need dynamic model in Mechanical Engineering
  - Dynamic modeling in robotic systems and controls
- Introduction to System Simulations
  - Static and dynamic simulations
  - General methods for system simulations
- Example
- Modeling and Simulation of Dynamic Systems in MuJoCo
  - Basic functions in MuJoCo
    - Developing XML model file;
    - Developing Python simulation controller;
  - System identification and optimization in MuJoCo

# What is Dynamic System?

**General defination from Webster Distionary:**

- **Dynamic**: A branch of mechanics that deals with forces and their relation primarily to the motion but sometimes also to the equilibrium of bodies;

- **System**: A regularly interacting or interdependent group of items forming a unified whole

**Our scope as of robotics and mechanical engineering:**

- **Dynamic:** Change of internal physical varaiables over time: force, velocity, pressure, fluid flow rate, current, voltage, temperature, etc.

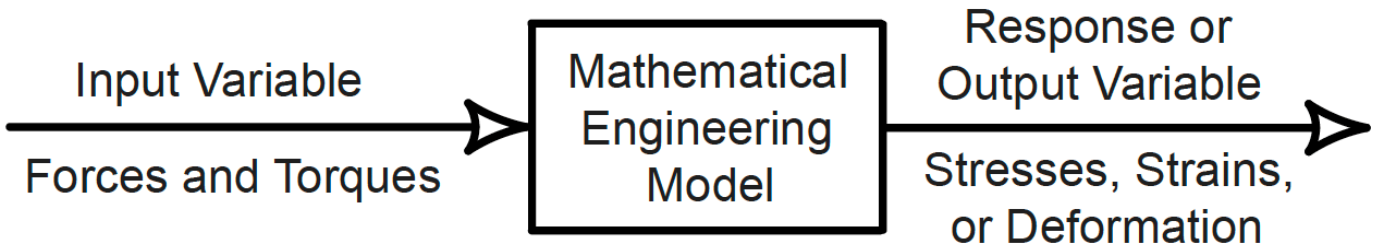
- **System:** robotic systems and the interacting environments (air, water, ground, human bodies, granular, gravity, magnetic, etc.)

# Why we need dynamic modeling?

Starship prototype reentry and landing


Champion-level Drone Racing

# Why we need dynamic modeling?



# How Dynamic Modeling can Help?

**Design**

- Simulate the designate motion, analyze the workspace, load distribution, verify your design;
- Optimize the design for better performance

**Control**

- Understand the responds from the system;
- Simulate and optimize the control law

**Machine Learning**

- Simulation, iterate to train the system

# General Steps for Dynamic System Modeling

1. **Define the system**
   Analyze the system's degrees of freedom, types of joints, locations, mass and inertias, end-effector's functions, etc.

2. **Develop kinematic equations: relation between joints and the end-effector**
   Forward kinematic: $\boldsymbol{\chi}_e = \boldsymbol{\chi}_e(\mathbf{q})$.
   Inverse kinematic: $\mathbf{q} = \mathbf{q}(\boldsymbol{\chi}_e^*)$

3. **Develop the equations of motion:** relationship between forces/torques and motion

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{b}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{J}_c(\mathbf{q})^T\,\mathbf{F}_c$$

4. **Solve the equations in time domain for analytical simulation**

# Introduction to System Simulation

**Goal of Static Simulation**

- Structure analysis;
- Stability analysis;
- Design validation

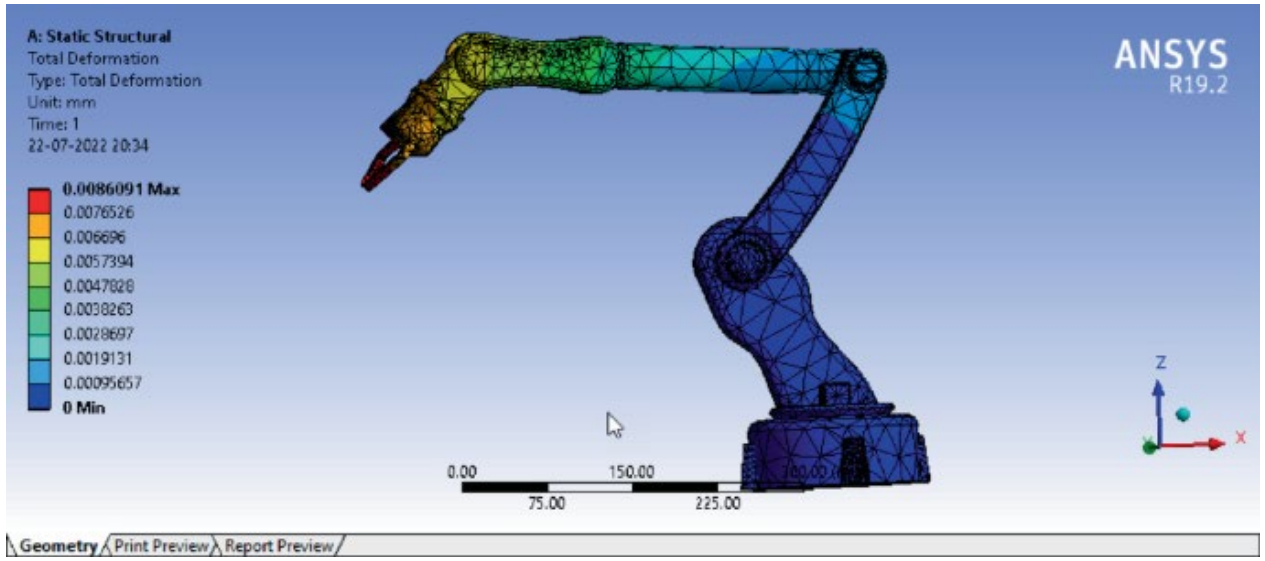
**Goal of Dynamic Simulation**

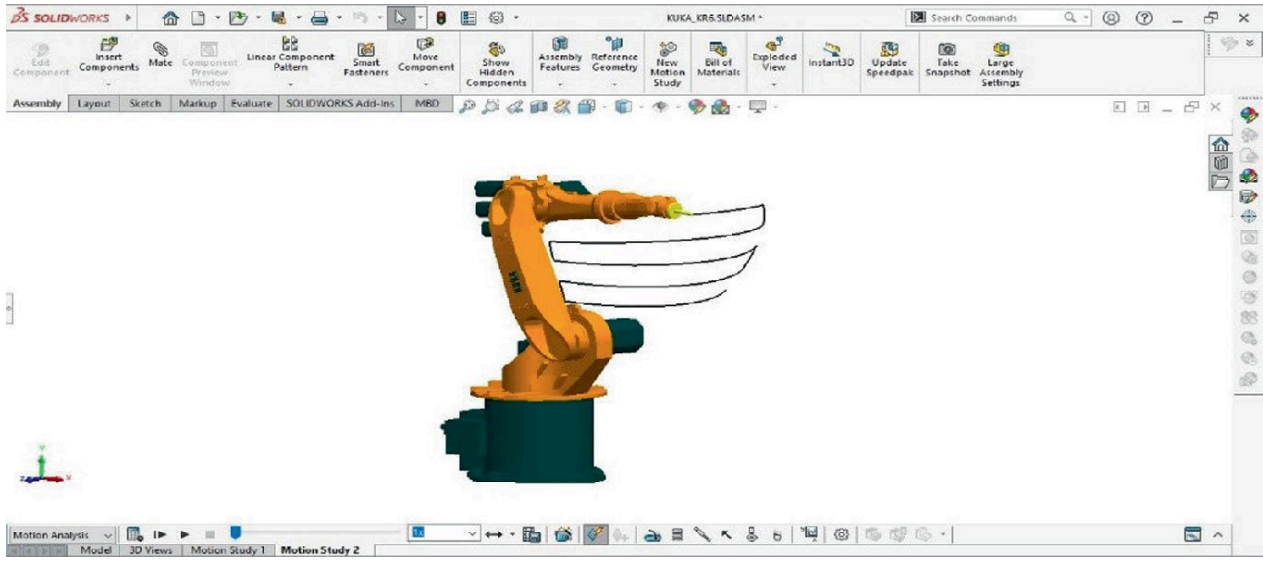
- Motion analysis;
- Control system design and validation;
- Trajectory and workspace planning;
- Optimization;
- Bridge to real-world task

# General methods for system simulations

**Static Simulation**

- **Mathematical Simulation**: solving mathematical models in Python, Matlab, etc.
- **CAD softwares:** Solidworks, Fusion 360, etc.
- **Finite Element Analysis(FEA) Softwares**: ANSYS, COMSOL, Abaqus, PyChrono, etc.



# General methods for system simulations

**Dynamic Simulation**

- **Mathematical Simulation**: solving equations of motion overtime in Python, Matlab Simulink, etc.
  - Pros: Fast, easily applied for simple systems;
  - Cons: Hard to apply on complex, non-linear systems;
- **Finite Element Analysis(FEA), Computational Fluid Dynamic(CFD), Fluid Structure Interaction(FSI) tools:** ANSYS, COMSOL, Abaqus, PyChrono, etc.
  - Pros: Commercial software, reliable, precise, friendly GUI, good for complex systems;
  - Cons: Commercial software, expensive, slow, hard to integrate to other functions.
- **Physics simulating libraries:** MuJoCo, PyBullet, Pynamics, etc.
  - Pros: Fast, acceptably precise, easy to integrate to other code/functions;
  - Cons: Steep learning curve

# Introduction to MuJoCo

**MuJoCo: Mu**lti-**Jo**int dynamics with **Co**ntact, a physics simulator developed by Google Deep mind

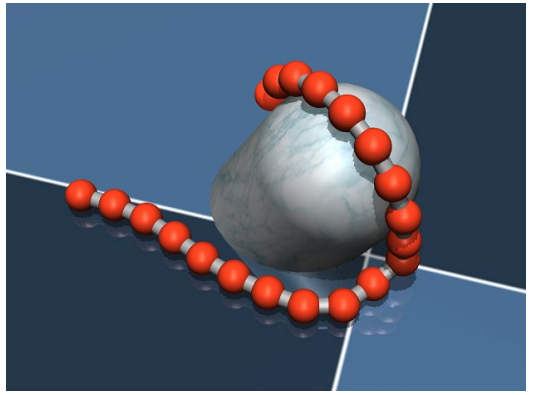
- C/C++ library with a C API;
- Python bindings;
- Unity plug-in
- GPU computation

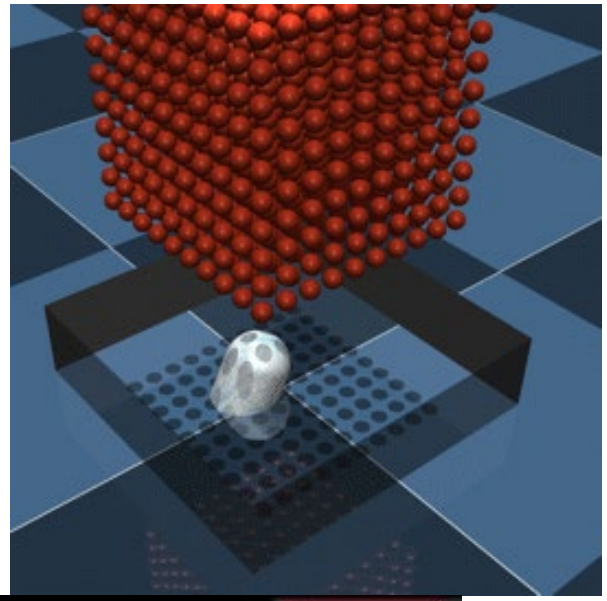
**Application:**

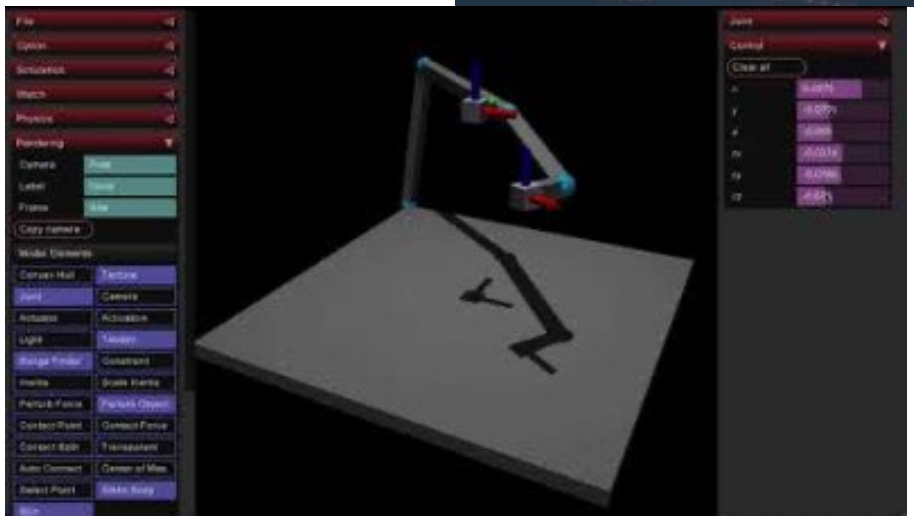
- Model-based computations such as control synthesis, state estimation, system identification, mechanism design, data analysis through inverse dynamics, and parallel sampling for machine learning applications.
- Traditional simulator, including for gaming and interactive virtual environments.

# Key Features of MuJoCo

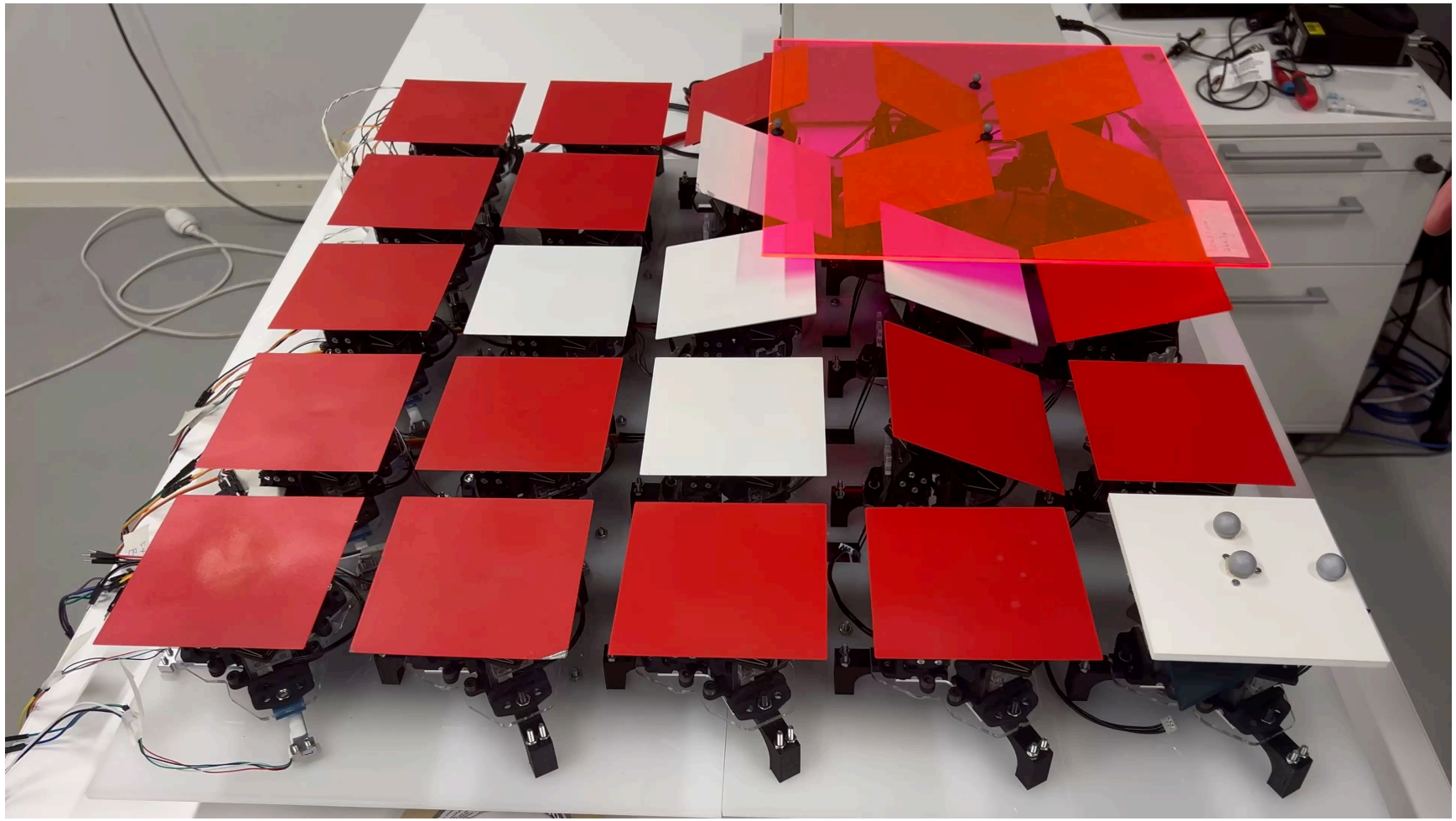
- **General actuation model**
  - Motors
  - Pneumatic and hydraulic cylinders,
  - PD controllers
  - Biological muscles
- **Soft, convex and analytically-invertible contact dynamics**
  - Interactions with various environments: ground, water, granular, etc.
- **Tendon geometry**
  - minimum-path-length strings obeying wrapping and via-point constraints
- **Reconfigurable computation pipeline**
  - Reconfigure your simulation on the fly using build-in flags
- **Interactive simulation and visualization**
  - 3D visualizer, easy for debugging and modeling







# Example: Control parameters optimization



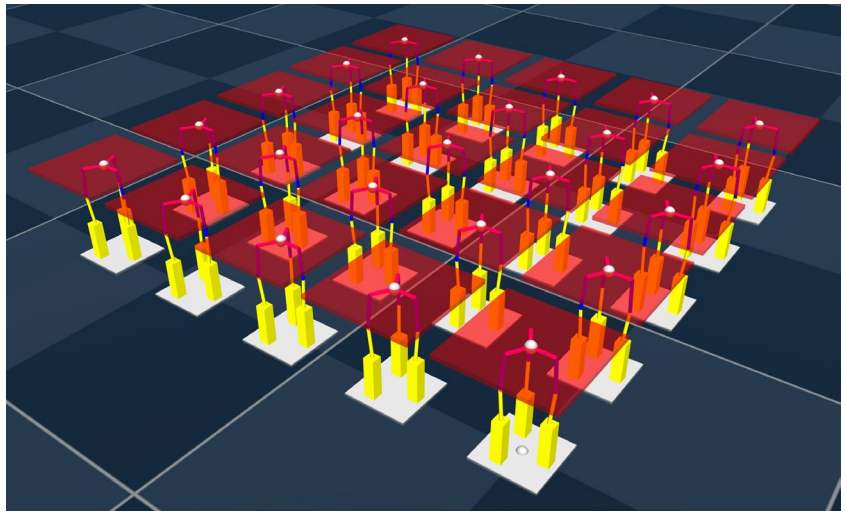
# Example: Control gait optimization

**Objective:**
Find optimal parameter set in formulas below to achieve fastest object moving speed

$$H(t) = h_{\text{amp}} \sin(2\pi f \cdot t + \phi) + h_0$$
$$\psi(t) = \psi_{\text{amp}} \sin(2\pi f \cdot t + \phi + \sigma) + \psi_0$$

**Search Space:**

| Parameter | Symbol | Search Space | Unit |
|---|---|---|---|
| Height amplitude | $h_{\text{amp}}$ | $[0.005, 0.04]$ | m |
| Inclination angle amplitude | $\psi_{\text{amp}}$ | $[0.35, 0.79]$ | radian |
| Frequency | $f$ | $[0.1, 0.8]$ | Hz |
| Resting height | $h_0$ | $[0.02, 0.04]$ | m |
| Resting inclination angle | $\psi_0$ | $[-0.26, 0.26]$ | radian |
| Height-inclination phase shift | $\phi$ | $[0, \pi]$ or $[\pi, 2\pi]$ | radian |
| Inter-group phase shift | $\delta$ | $[0, 2\pi]$ | radian |
| Tile contact threshold | $\epsilon$ | $[0.1, 0.5]$ | - |



# Example: Control gait optimization

**How to get it done:**

1. Run mujoco simulation using the suggested parameters from optimizer;

2. Evaluate the speed of the object;

3. Send it back to optimizer for next iterations;

# Example: Control gait optimization

**Why it is good?**

1. Efficient: no prototyping test needed;

2. Low cost: different objects can be explored, no manufacturing;

3. Generalized solution: you can apply to any other scenarios that requires optimization;

**Why it is not good?**

1. Sim to real gap: require further calibrations;

2. Not easy for beginners;

3. Overkill if the optimization problem is easy to solve;

# MuJoCo File Structure

It takes at least two files to run and control the simulation in MuJoCo:

- XML model file for model;
- Python code for controlling the simulation;
- Simulation asset files (optional)

Model should be developed as a collection of rigid bodies with joints linked together in a kinematic tree/chain manner.

# Modeling in XML File

**General body properties:**

- **Position:** real(3)
- **Geom**
  - Type: plane, sphere, capsule, ellipsoid, cylinder, box, mesh
  - Size: real(2) or real(3)
  - Mesh: optional, if you want to import your own stl file
- **Frame orientations:** euler real(3)
- **Inertial**
  - mass: real(3)
  - diaginertia: real(3)
  - pos: real(3)

- **Joint:**
  - Type: free, ball, slide, hinge
  - Pos: real(3)
  - Axis: real(3)
  - Stiffness: real
  - Damping: real
  - Range: real(2)
  - Frictionloss: real

# Modeling in XML File

**General actuator properties:**

- **type**:
  - Motor: torque;
  - position;
  - velocity;
  - damper;
  - cylinder: pneumatic or hydraulic cylinders);
  - muscle;
  - adhesion: injects forces at contacts in the normal direction
  - …
- **site/joint**
- **Forcerange**: real(2)
- **gain**

# Modeling in XML File

**General sensor properties:**

- **type**:
  - touch: detect contact;
  - accelerometer;
  - velocimeter;
  - gyro;
  - force;
  - torque;
  - magnetometer;
  - …
- **site/joint**
- **noise**

# Run simulation in Python

**Import library:**

```
import mujoco
```

**Read xml file:**

```
model = mujoco.MjModel.from_xml_path(xml_path)
data = mujoco.MjData(model)
```

**Set a controller call back:**

```
mujoco.set_mjcb_control(mycontroller)
```

**Run a step of simulation:**

```
mujoco.mj_step(model, data)
```

**Read body position data:**

```
main_body_pos = data.body("main_body").xpos
```

**Read sensor data:**

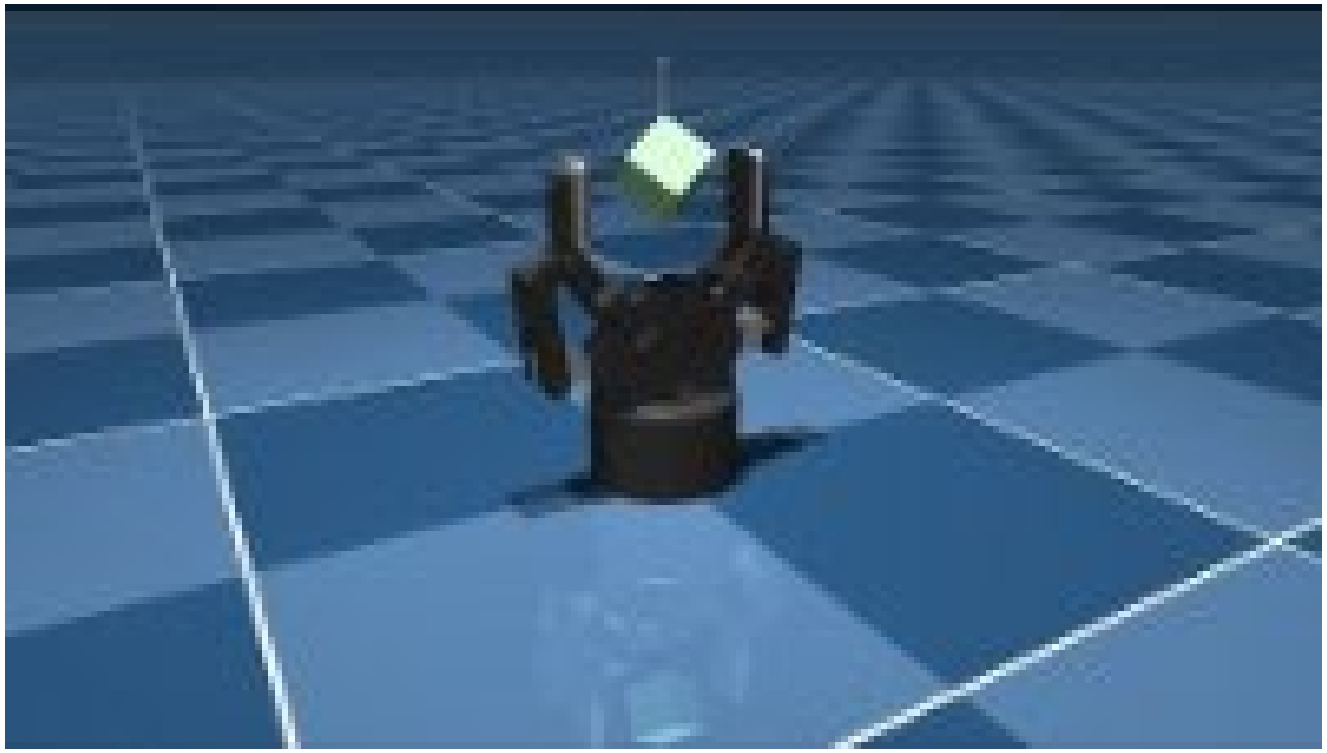
```
data = data.sensor('SENSOR_NAME').data
```

**Controller callback function:**

```
def mycontroller (data, vel):
    t = data.time
    goal_pos = vel*t
    data.ctrl[0] = goal_pos /180*np.pi
    data.ctrl[1] = goal_pos /180*np.pi
    data.ctrl[2] = goal_pos /180*np.pi
    return
```

# Demo: robotiq_2f85

Source: https://github.com/google-deepmind/mujoco_menagerie/tree/main/robotiq_2f85



# Learning Resources

- MuJoCo official document: https://mujoco.readthedocs.io/en/latest/overview.html
- Online course: https://pab47.github.io/mujoco.html
- Robot dynamics and simulation Lecture Notes, Allison Okamura, Stanford University: https://web.stanford.edu/class/me328/lectures/lecture5-dynamics.pdf
- Robot Dynamics Lecture Notes, Robotic Systems Lab, ETH Zurich: https://ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/documents/RobotDynamics2017/RD_HS2017script.pdf
- Handbook of Robotics: https://link.springer.com/book/10.1007/978-3-540-30301-5
- Robotics Modelling, Planning and Control: https://link.springer.com/book/10.1007/978-1-84628-642-1

# Questions?