

# A Practical Isaac Sim Tutorial



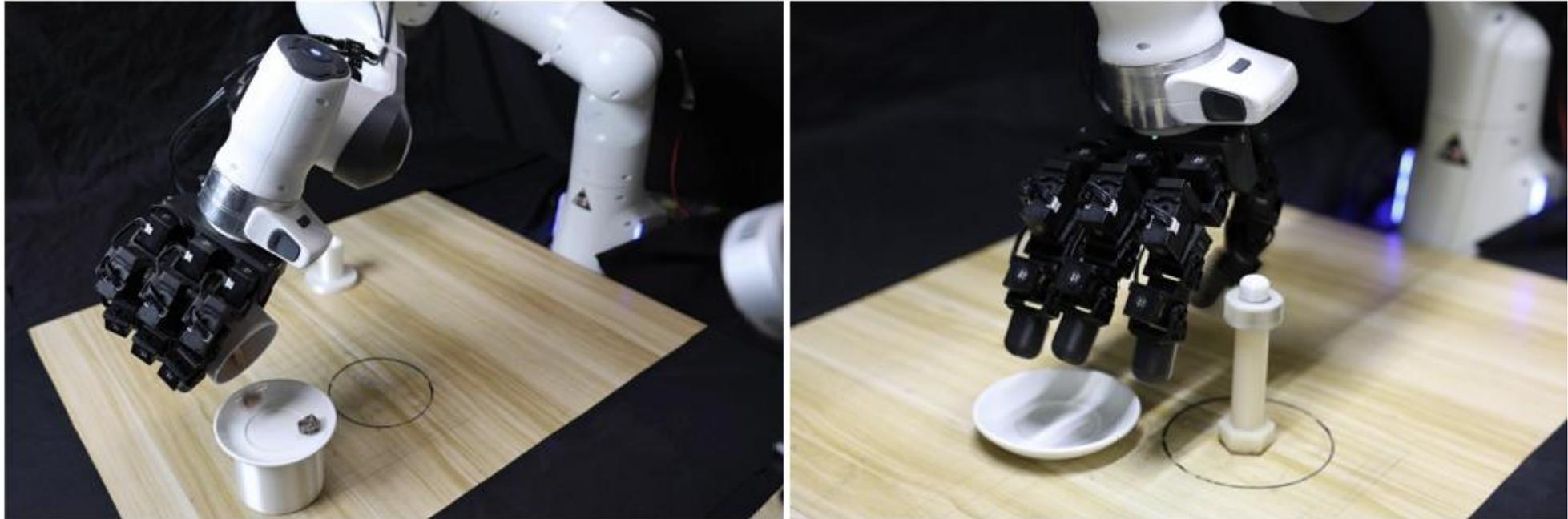
# Outline

**Background**

**High-level  
Structure of  
Issac Sim**

**The API of  
Issac Sim**

# Real-World Robotics Is Hard to Scale



## Drawback of Real-world Robotics

Speed: **Slow**

Expenses: **Expensive**

Robustness: **Fragile**

# An Alternative Solution – CPU Simulator

## The Design Purpose of CPU Simulator:

Provide a **faster**, **cheaper**, and **safer** environment than real-world robotics, enabling rapid experimentation, debugging, and early-stage development without hardware risks.

MJ

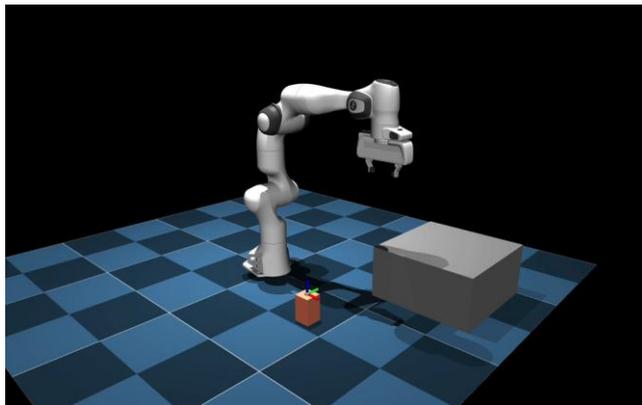
## MuJoCo

Fast, accurate rigid-body dynamics with smooth contact handling. Widely used in RL research.

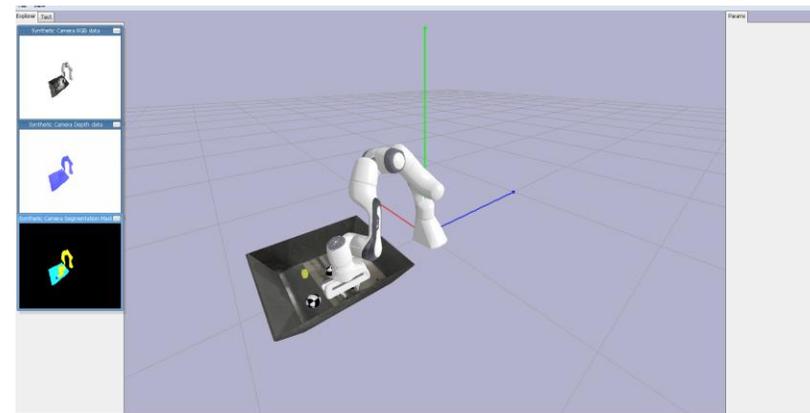
PB

## PyBullet

Open-source physics engine with Python bindings. Easy to start. Popular for manipulation research.

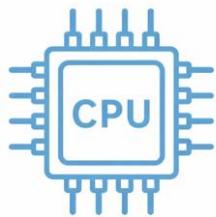


MuJoCo



PyBullet

# Why choose GPU Simulator



## CPU Simulator

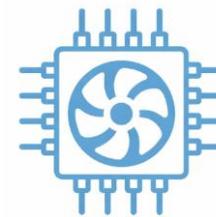
**Speed:** MuJoCo (~10M steps/sec)

**Parallelism:** Large-scale parallelism is limited by CPU cores

**Rendering:** Rely on simpler or external rendering pipelines

**Physics Fidelity:** Strong in stability and controllability

**Accessibility:** Easy to run



## GPU Simulator

**Speed:** Issac Sim (~1B steps/sec)

**Processes:** Designed for massively parallel

**Rendering:** Real-time, photorealistic rendering

**Physics Fidelity :** Depend heavily on engine configuration

**Accessibility:** Require GPUs

CPU simulators are often preferred for accessibility and traditional robotics workflows, while GPU simulators excel at massively parallel RL and high-quality real-time rendering—yet neither automatically implies differentiable physics.

# Outline

**Background**

**High-level  
Structure of  
Issac Sim**

**The API of  
Issac Sim**

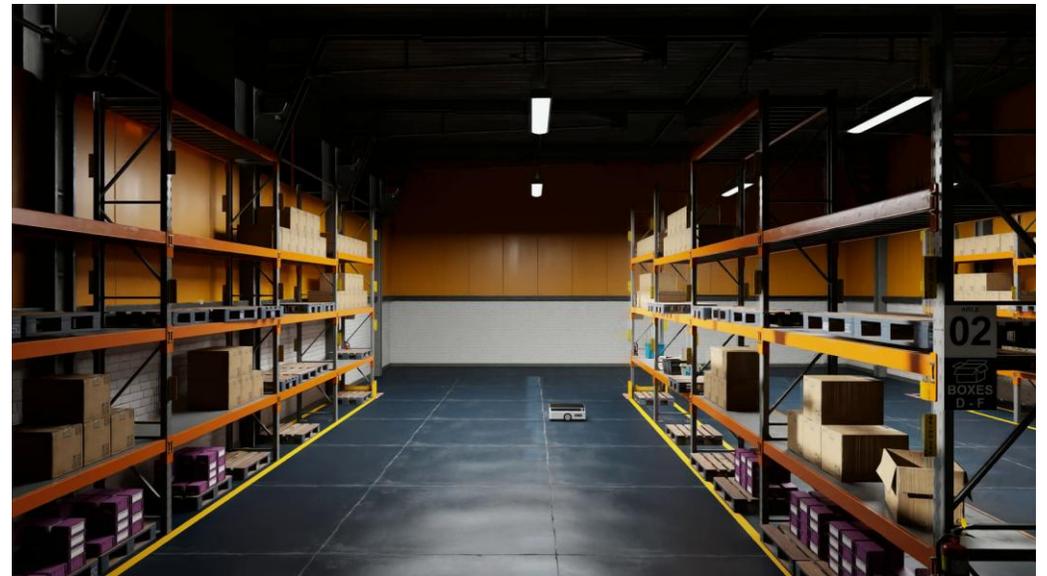
# Isaac Sim: A Flagship GPU-Based Robotics Simulator

## The Design Purpose of Isaac Sim:

Leverage GPU acceleration to enable **large-scale, highly parallel** robotics simulation for reinforcement learning, synthetic data generation, and rapid experimentation.



Highly Parallel



Photorealistic Rendering

# High-Level Isaac Sim Architecture

## **USD defines scenes and robots:**

All environments, robots, sensors, and assets are represented using USD, providing a unified scene description.

## **GPU physics runs dynamics:**

Physics simulation is executed on the GPU, enabling fast dynamics computation and efficient large-scale simulation.

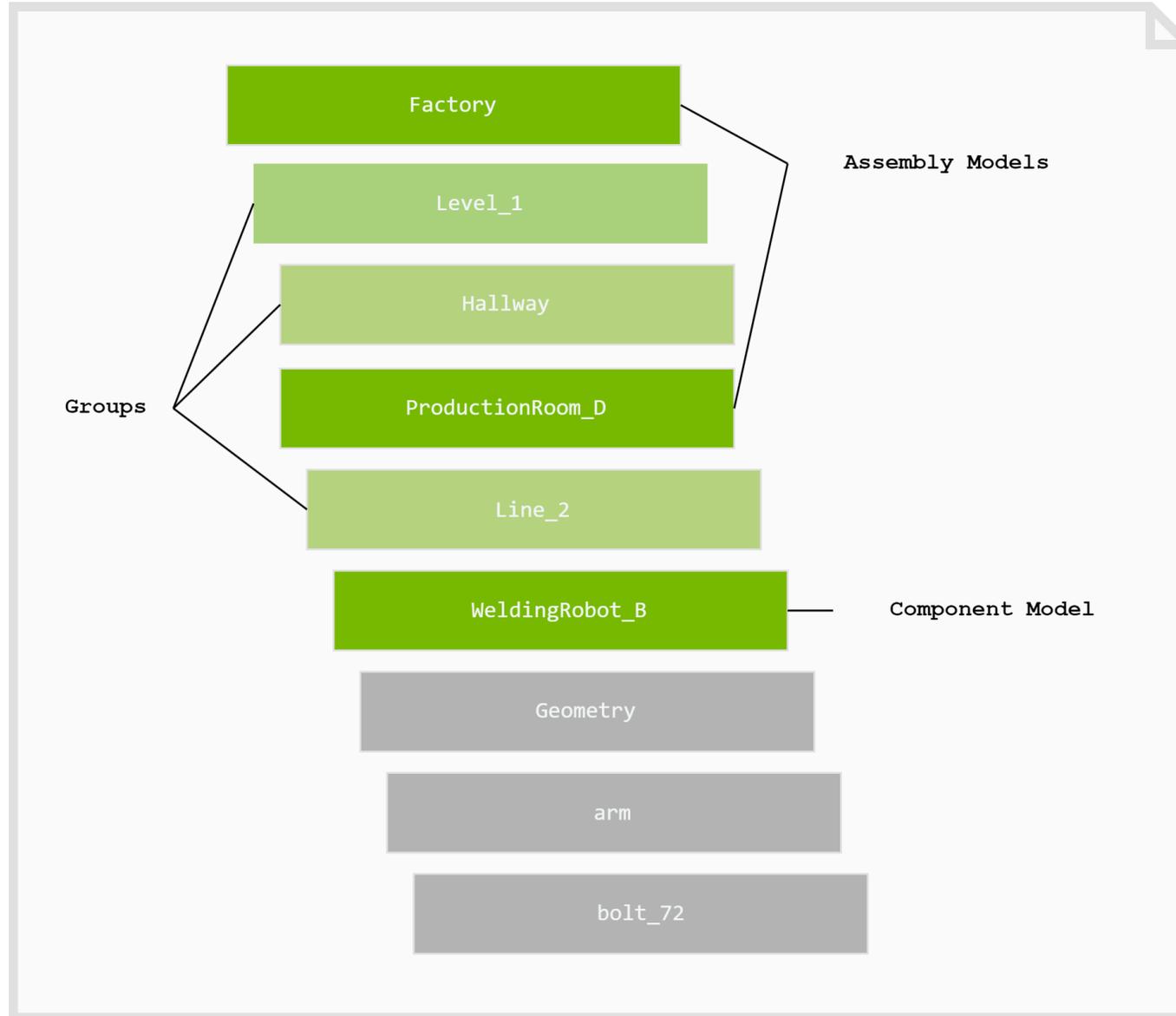
## **Environments are batched:**

Multiple simulation environments are batched and executed in parallel, which is essential for high-throughput reinforcement learning.

## **GUI and Python control the same world:**

The graphical interface and Python APIs operate on the same underlying simulation state, allowing seamless switching between visual debugging and programmatic control.

# USD Structure



# An Example for Issac Sim GUI

The screenshot shows a web browser displaying the Isaac Sim documentation page. The page content includes a step-by-step guide for connecting nodes in a graph. A terminal window is overlaid on the page, showing the execution of the Isaac Sim command line interface. The terminal output displays various startup messages and warnings, including a warning about a crash reporter and another about OmniHub accessibility. The background of the terminal shows a desert landscape.

docs.isaacsim.omniverse.nvidia.com/5.1.0/omniagraph/o...

Isaac Sim Documentation

3. Connect the `Tick` output of the `On Playback Tick` node to the `Exec In` input of both controller

```
65\physx\source\physx\src\NpMaterial.cpp, LINE 186
2026-02-05 13:50:23 [199,073ms] [Warning] [omni.physx.plugin] Detected an articulation at /World/jetbot with more than 4 v
elocity iterations being added to a TGS scene.The related behavior changed recently, please consult the changelog. This wa
rning will only print once.

(isaaclab) D:\Internship\2025_Spring\IsaacLab>isaacsim
Loading user config located at: 'C:/Users/lvwen/AppData/Local/ov/data/Kit/Isaac-Sim Full/4.5/user.config.json'
[Info] [carb] Logging to file: C:/Users/lvwen/.nvidia-omniverse/logs/Kit/Isaac-Sim Full/4.5/kit_20260205_215353.log
2026-02-05 13:53:53 [0ms] [Warning] [omni.kit.app.plugin] No crash reporter present, dumps uploading isn't available.
[0.410s] [ext: omni.kit.async_engine-0.0.1] startup
[1.417s] [ext: omni.metrics.core-0.0.1] startup
[1.422s] [ext: omni.client.lib-1.0.0] startup
[1.498s] [ext: omni.blobkey-1.1.2] startup
[1.503s] [ext: omni.stats-1.0.1] startup
[1.512s] [ext: omni.datastore-0.0.0] startup
[1.522s] [ext: omni.client-1.2.2] startup
[1.543s] [ext: omni.ujitso.default-1.0.0] startup
[1.553s] [ext: omni.hsscclient-1.1.1] startup
2026-02-05 13:53:54 [1,391ms] [Warning] [omni.datastore] OmniHub is inaccessible
[1.580s] [ext: omni.gpu_foundation.shadercache.vulkan-1.0.0] startup
[1.600s] [ext: omni.assets.plugins-0.0.0] startup
[1.612s] [ext: omni.gpu_foundation-0.0.0] startup
[1.701s] [ext: carb.windowing.plugins-1.0.0] startup
[1.856s] [ext: omni.kit.renderer.init-0.0.0] startup
[1.861s] [ext: omni.kit.pipapi-0.0.0] startup
[1.959s] [ext: omni.materialx.libs-1.0.6] startup
[2.006s] [ext: omni.kit.pip_archive-0.0.0] startup
[2.008s] [ext: omni.usd.config-1.0.5] startup
[2.085s] [ext: omni.gpucompute.plugins-0.0.0] startup
```

Simple differential control for the JetBot

1. Press the play button.

2. Select the `Differential Controller` node in the graph.

2 KB  
11 KB  
3 KB  
7 KB

00:04

搜索

4 -3°C 晴明 21:53 2026/2/5

# Outline

**Background**

**High-level  
Structure of  
Issac Sim**

**The API of  
Issac Sim**

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 0 — Launching Isaac Sim and Initializing the App

```
from isaacsim.app import AppLauncher

parser = argparse.ArgumentParser(description="Tutorial on using the differential IK controller.")
parser.add_argument("--robot", type=str, default="franka_panda")
parser.add_argument("--num_envs", type=int, default=128)

AppLauncher.add_app_launcher_args(parser)
args_cli = parser.parse_args()

app_launcher = AppLauncher(args_cli)
simulation_app = app_launcher.app
```

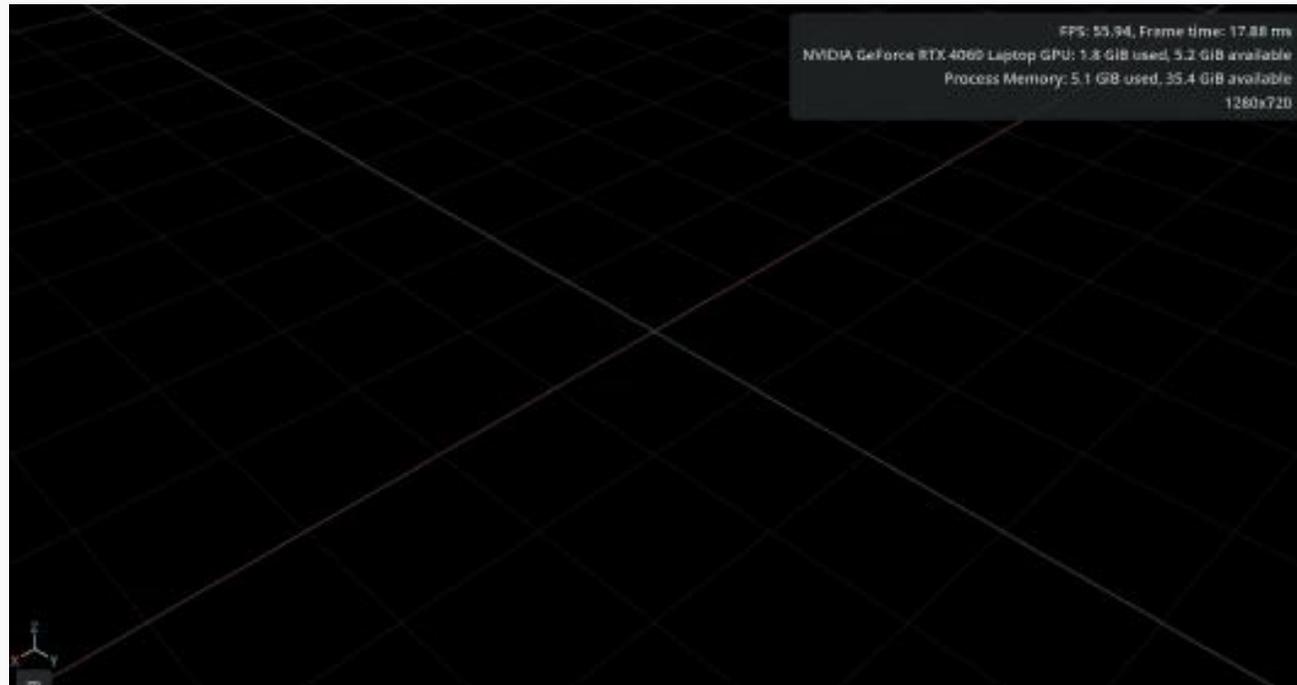
**add\_app\_launcher\_args** helps us to add basic Isaac Sim argument like the device and the type of renderer.

**Instantiate APPLauncher** to create the environments.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 0 — Launching Isaac Sim and Initializing the App



`add_app_launcher_args` helps us to add basic Issac Sim argument like the device and the type of renderer.

**Instantiate APPLauncher** to create the environments.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 1 — Defining the Scene Structure with Configuration

```
@configclass
class TableTopSceneCfg(InteractiveSceneCfg):
    ground = AssetBaseCfg(
        prim_path="/World/defaultGroundPlane",
        spawn=sim_utils.GroundPlaneCfg(),
        init_state=AssetBaseCfg.InitialStateCfg(pos=(0.0, 0.0, -1.05)),
    )

    dome_light = AssetBaseCfg(
        prim_path="/World/Light",
        spawn=sim_utils.DomeLightCfg(intensity=3000.0, color=(0.75, 0.75, 0.75)),
    )

    table = AssetBaseCfg(
        prim_path="{ENV_REGEX_NS}/Table",
        spawn=sim_utils.UsdFileCfg(
            usd_path=f"{ISAAC_NUCLEUS_DIR}/Props/Mounts/Stand/stand_instanceable.usd",
            scale=(2.0, 2.0, 2.0),
        ),
    )
```

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 2 — Selecting and Instantiating Robot Assets (in TableTopSceneConfig)

```
from isaacsim_assets import FRANKA_PANDA_HIGH_PD_CFG, UR10_CFG

if args_cli.robot == "franka_panda":
    robot = FRANKA_PANDA_HIGH_PD_CFG.replace(prim_path="{ENV_REGEX_NS}/Robot")
elif args_cli.robot == "ur10":
    robot = UR10_CFG.replace(prim_path="{ENV_REGEX_NS}/Robot")
else:
    raise ValueError("Unsupported robot")
```

We define all the assets (ground, light, table and robot) in **TableTopSceneConfig**.

We don't need to define config for each environment separately.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 3 — Creating the Simulation Context and Scene

```
sim_cfg = sim_utils.SimulationCfg(dt=0.01, device=args_cli.device)
sim = sim_utils.SimulationContext(sim_cfg)

sim.set_camera_view([2.5, 2.5, 2.5], [0.0, 0.0, 0.0])

scene_cfg = TableTopSceneCfg(num_envs=args_cli.num_envs, env_spacing=2.0)
scene = InteractiveScene(scene_cfg)

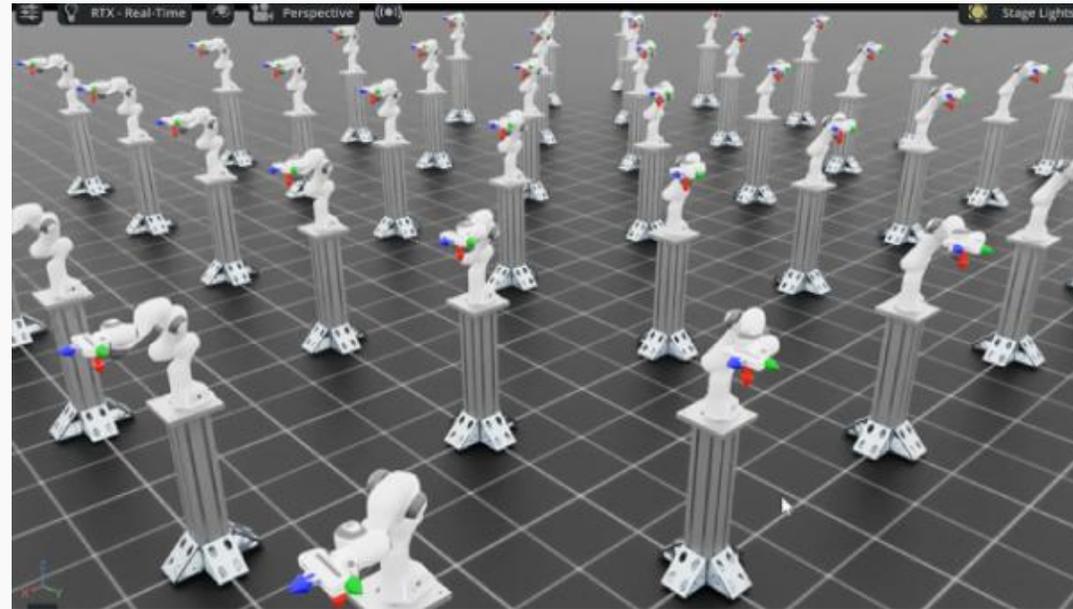
sim.reset()
```

We feed the **TableTopSceneConfig** to `InteractiveScene`, and create the simulation context.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 3 — Creating the Simulation Context and Scene



We feed the **TableTopSceneConfig** to InteractiveScene, and create the simulation context.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 4 — Accessing Scene Entities and Creating Controllers

```
robot = scene["robot"]

from isaacsim.controllers import DifferentialIKController, DifferentialIKControllerCfg
diff_ik_cfg = DifferentialIKControllerCfg(command_type="pose", use_relative_mode=False, ik_method
diff_ik_controller = DifferentialIKController(diff_ik_cfg, num_envs=scene.num_envs, device=sim.de
```

Isaac Sim has already implemented some IK algorithms, we can use them with **DifferentialIKControllerCFG** and **DifferentialIKController**. Please check the specific API in their document.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 5 — Defining Batched Goals for Parallel Environments

```
ee_goals = torch.tensor([
    [0.5, 0.5, 0.7, 0.707, 0, 0.707, 0],
    [0.5, -0.4, 0.6, 0.707, 0.707, 0.0, 0.0],
    [0.5, 0, 0.5, 0.0, 1.0, 0.0, 0.0],
], device=sim.device)

current_goal_idx = 0
ik_commands = torch.zeros(scene.num_envs, diff_ik_controller.action_dim, device=robot.device)
ik_commands[:] = ee_goals[current_goal_idx]
```

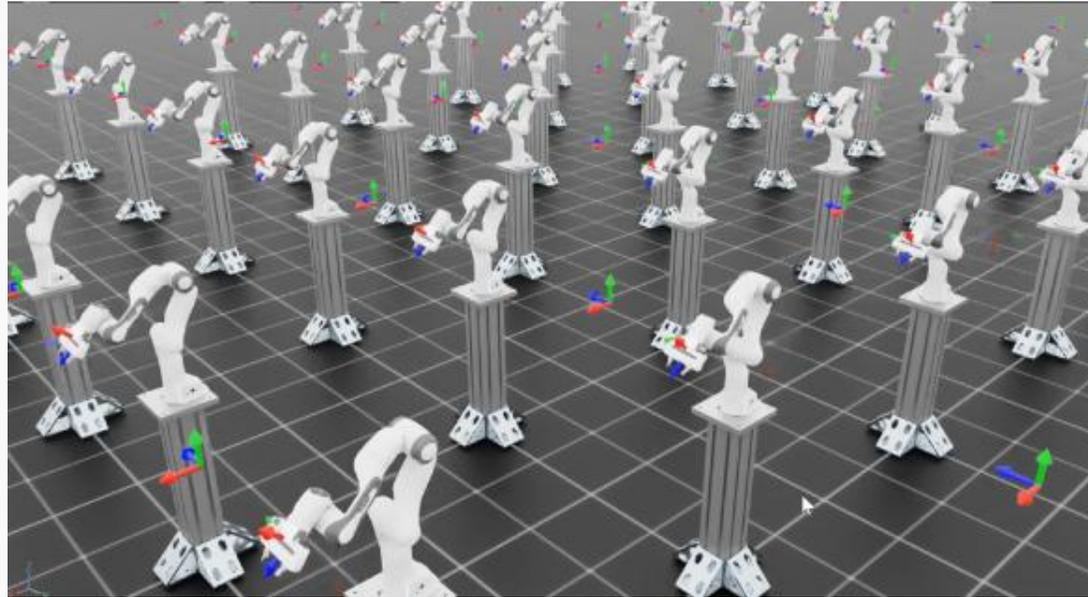
The input of the environments is batched. Depends on our requirements, we can control them separately or synchronously.

The goals are the quaternions and coordinates of 3 fingertips.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 5 — Defining Batched Goals for Parallel Environments



The input of the environments is batched. Depends on our requirements, we can control them separately or synchronously.

The goals are the quaternions and coordinates of 3 fingertips.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 6 — Simulation Loop: Reset

```
if count % 150 == 0:
    joint_pos = robot.data.default_joint_pos.clone()
    joint_vel = robot.data.default_joint_vel.clone()
    robot.write_joint_state_to_sim(joint_pos, joint_vel)
    robot.reset()

    ik_commands[:] = ee_goals[current_goal_idx]
    joint_pos_des = joint_pos[:, robot_entity_cfg.joint_ids].clone()

    diff_ik_controller.reset()
    diff_ik_controller.set_command(ik_commands)

    current_goal_idx = (current_goal_idx + 1) % len(ee_goals)
```

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 6 — Simulation Loop: Observe

```
jacobian = robot.root_physx_view.get_jacobians()[:, ee_jacobi_idx, :, robot_entity_cfg.joint_ids]
ee_pose_w = robot.data.body_pose_w[:, robot_entity_cfg.body_ids[0]]
root_pose_w = robot.data.root_pose_w
joint_pos = robot.data.joint_pos[:, robot_entity_cfg.joint_ids]
```

The design of observation is important based on your task. You can get both the joint state and rendered images with Issac Gym.

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 6 — Simulation Loop: Compute

```
ee_pos_b, ee_quat_b = subtract_frame_transforms(  
    root_pose_w[:, 0:3], root_pose_w[:, 3:7],  
    ee_pose_w[:, 0:3], ee_pose_w[:, 3:7],  
)  
  
joint_pos_des = diff_ik_controller.compute(ee_pos_b, ee_quat_b, jacobian, joint_pos)
```

# Official Example for Issac Sim (Inverse Kinematic)

[https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05\\_controllers/run\\_diff\\_ik.html](https://isaac-sim.github.io/IsaacLab/main/source/tutorials/05_controllers/run_diff_ik.html)

## Step 6 — Simulation Loop: Act

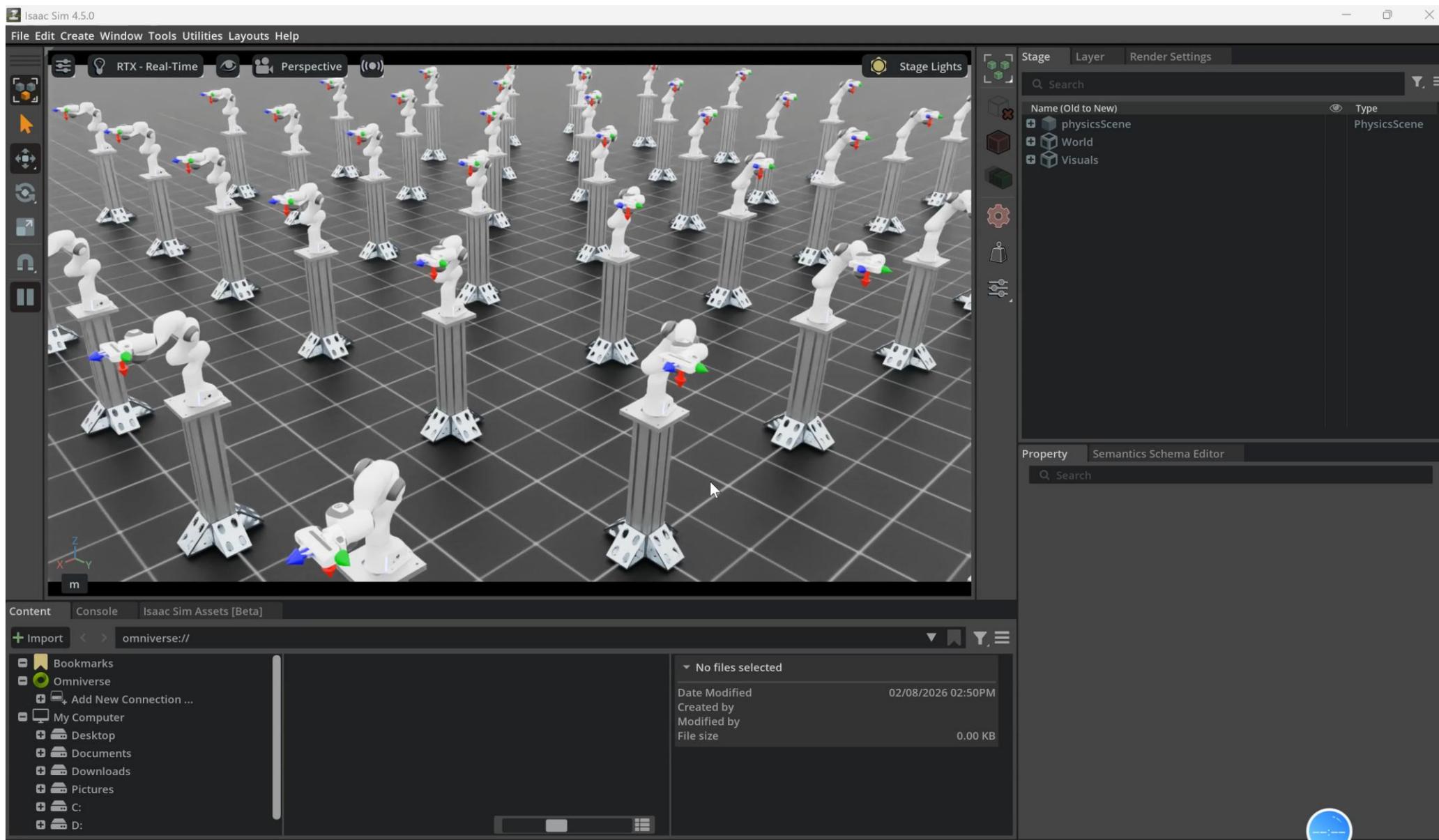
```
robot.set_joint_position_target(joint_pos_des, joint_ids=robot_entity_cfg.joint_ids)
scene.write_data_to_sim()

sim.step()

scene.update(sim_dt)
count += 1
```

Interactively apply the Inverse Kinematic, we can move the end-effector to the desired position.

# Official Example for Issac Sim (Inverse Kinematic)



Q&A