RIGGING ROBOTS IN OMNIVERSE

ELA NUÑEZ

SUMMER 2024

NAVAL RESEARCH LABORATORY HBCU/MI INTERN

IMITATIVE LEARNING

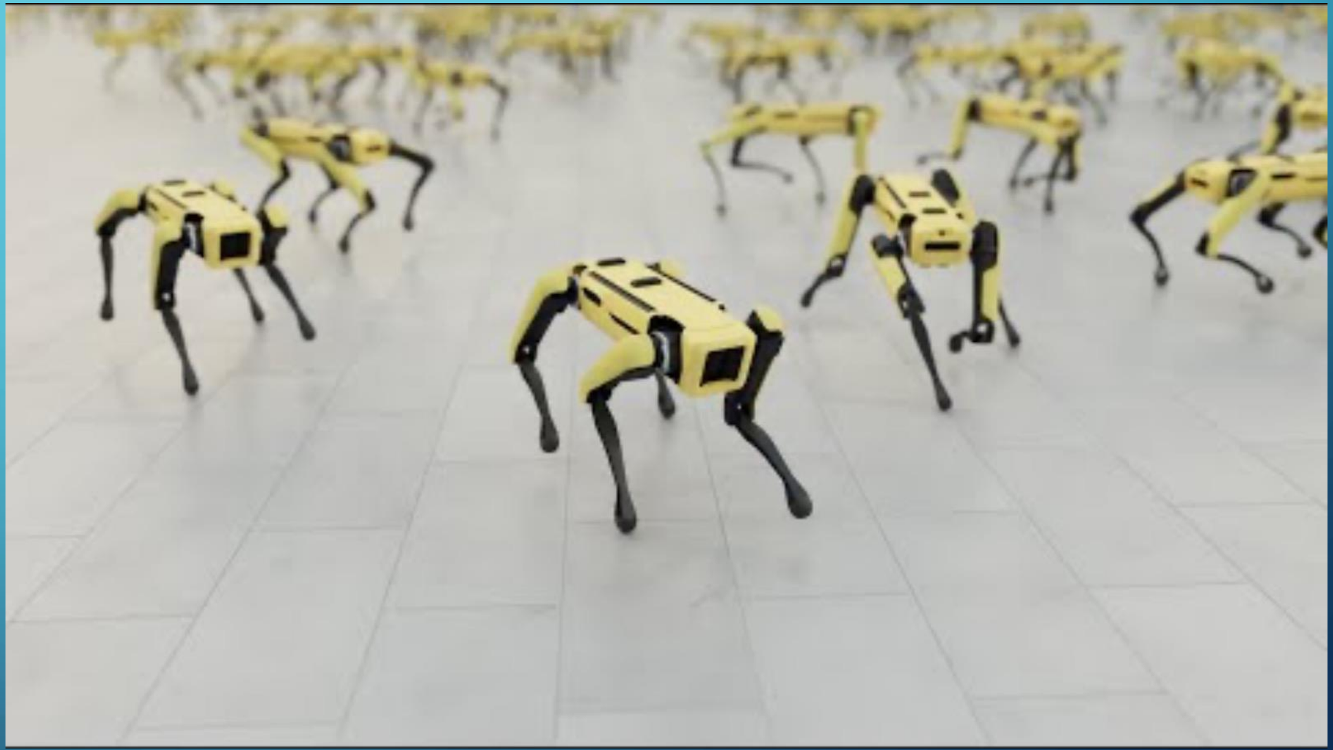
- Training robots
- Expert data
- Rewards for autonomy

https://ai.stanford.edu/blog/learning-to-imitate/
https://mobile-aloha.github.io/

PART 1: SETUP

NVIDIA ISAAC-SIM OMNIVERSE

- Simulation software
- Fine-tuning
- Efficient



https://www.nvidia.com/en-us/omniverse/
https://developer.nvidia.com/blog/closing-the-sim-to-real-gap-training-spot-quadruped-locomotion-with-nvidia-isaac-lab/

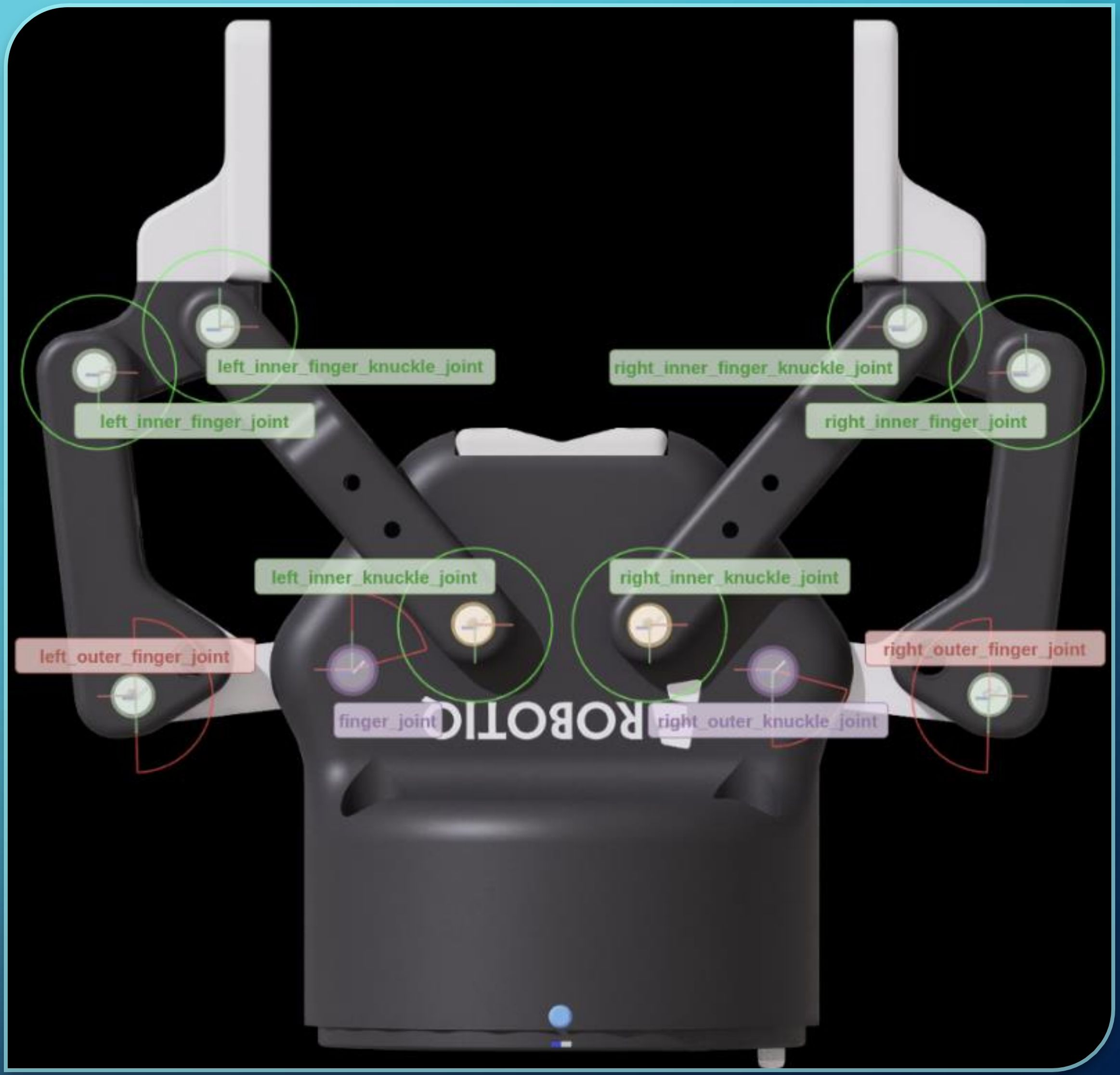
ROBOTIQ 2F85 GRIPPER

- Kinova G3 arm
- Position, velocity, and force control
- finger_joint and right_outer_knuckle_joint

https://robotiq.com/products/2f85-140-adaptive-robot-gripper?ref=nav_product_new_button
https://blog.robotiq.com/the-new-fanuc-crx-cobot-now-supports-robotiq-grippers

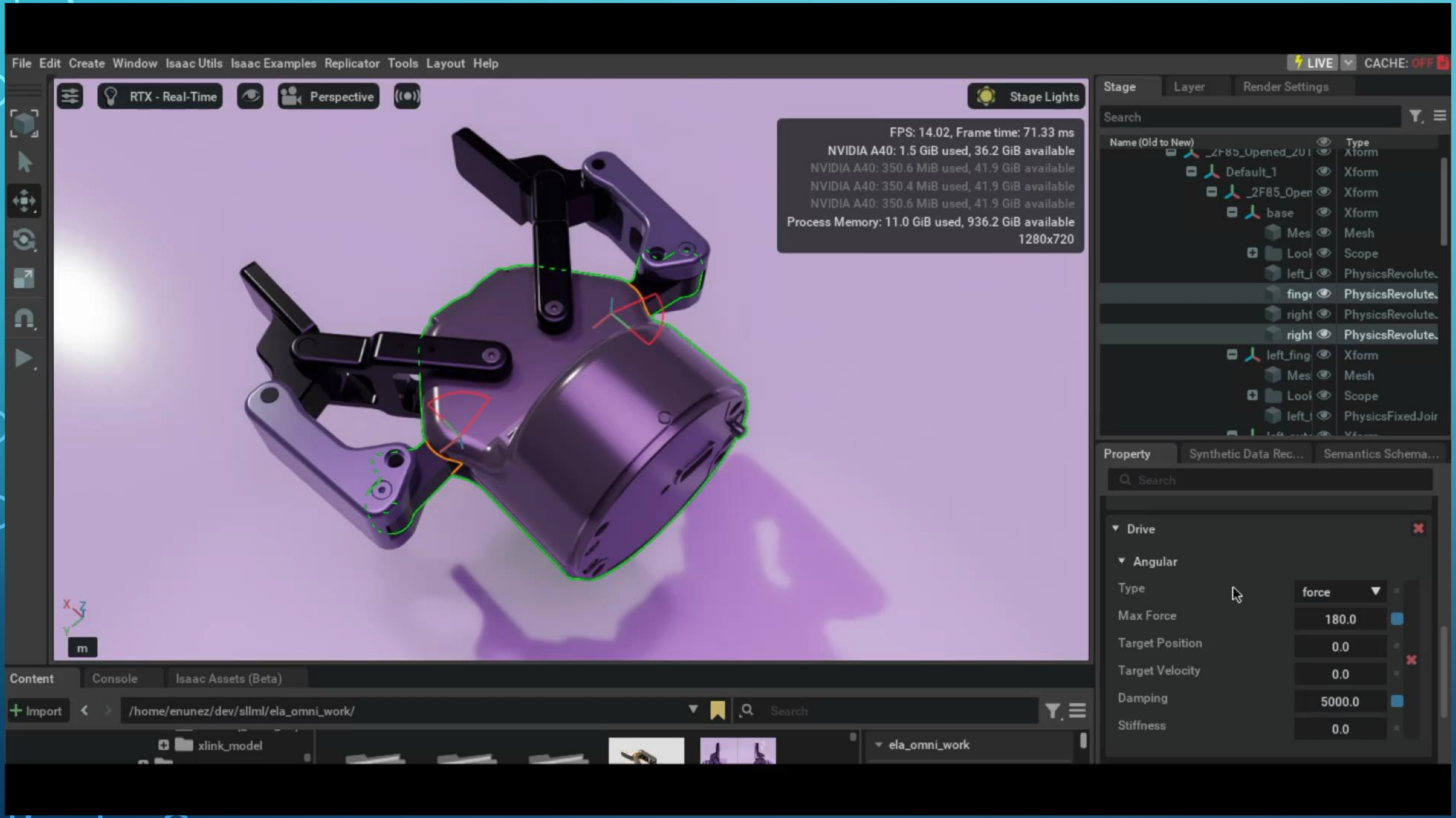
METHODS

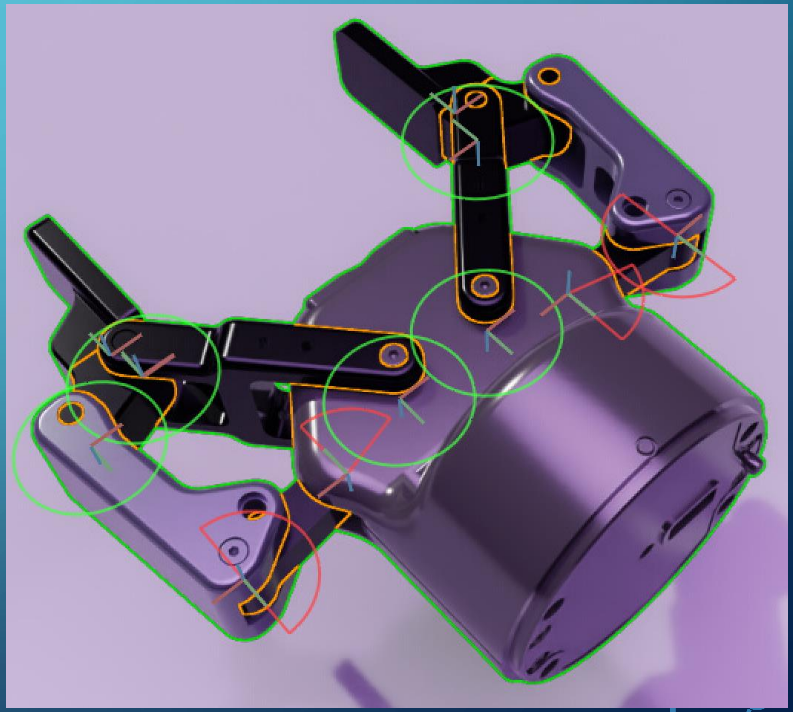
- "Rigging complex structures" tutorial
  - Import CAD
  - Add joints
  - Break articulation loop
  - Add joint drives, colliders, and mimic joint
  - Create simulation environment
- 1 change at a time
- Troubleshooting: keywords, Isaac-Sim forums/docs, blogs, A.I., YouTube, team

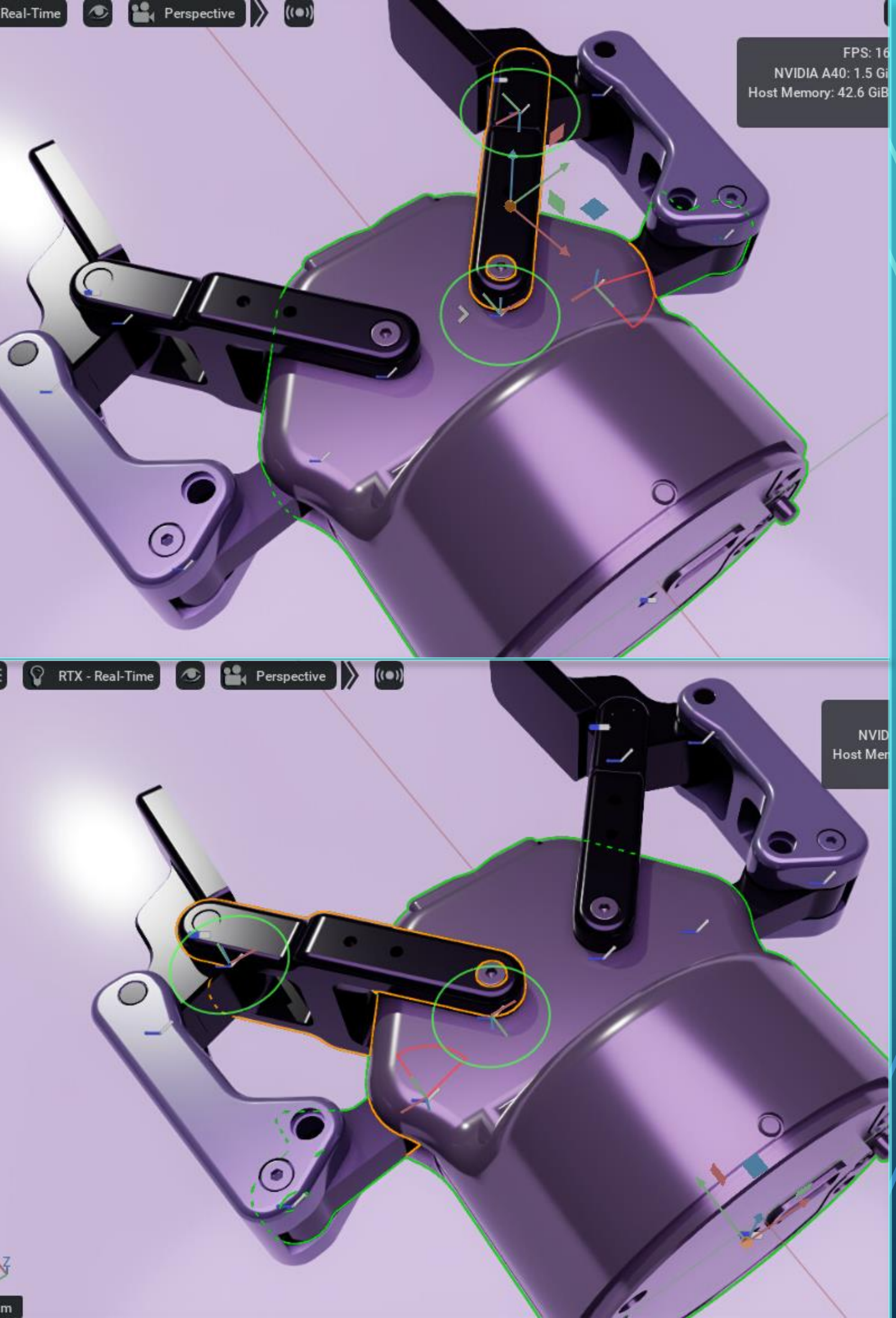


https://docs.omniverse.nvidia.com/isaacsim/latest/advanced_tutorials/tutorial_advanced_rigging_complex_structures.html

RESULTS


File Edit Create Window Isaac Utils Isaac Examples Replicator Tools Layout Help
LIVE
CACHE: OFF
RTX - Real-Time
Perspective
Stage Lights
FPS: 14.02, Frame time: 71.33 ms
NVIDIA A40: 1.5 GiB used, 36.2 GiB available
NVIDIA A40: 350.6 MiB used, 41.9 GiB available
NVIDIA A40: 350.4 MiB used, 41.9 GiB available
NVIDIA A40: 350.6 MiB used, 41.9 GiB available
Process Memory: 11.0 GiB used, 936.2 GiB available
1280x720
Stage
Layer
Render Settings
Search
Name (Old to New)
Type
Default_1
Xform
base
Mesh
Scope
PhysicsRevolute
PhysicsFixedJoir
Property
Synthetic Data Rec...
Semantics Schema...
Search
Drive
Angular
Type
force
Max Force
180.0
Target Position
0.0
Target Velocity
0.0
Damping
5000.0
Stiffness
0.0
Content
Console
Isaac Assets (Beta)
Import
/home/enunez/dev/sllml/ela_omni_work/
xlink_model
ela_omni_work





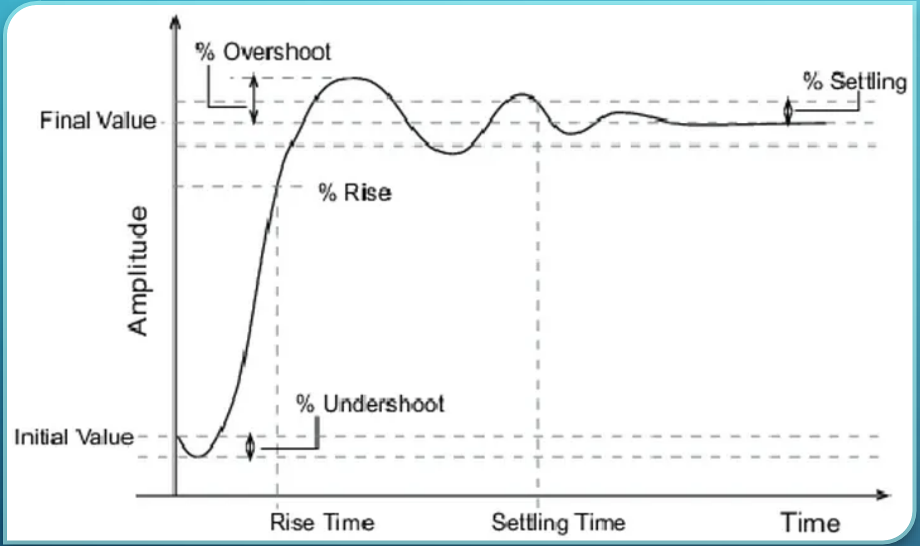
CONCLUSIONS

- Broken right finger
- Joint articulation = tree
- Iterations need time
- Damping controls Δv, stiffness controls Δd
- Replaceable fingertips!
- Update software

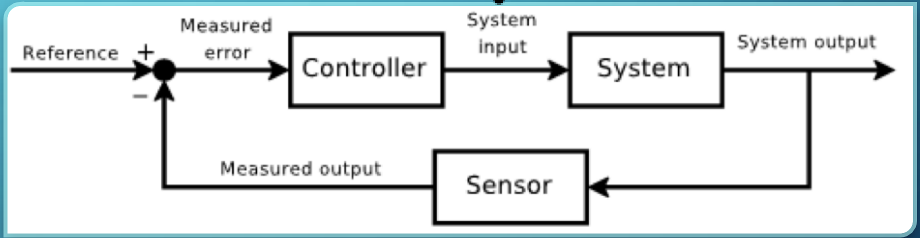
PART 2: CONTROL

CONTROL THEORY

PD controller: control_effort = self.kp * pos_error + self.kd * vel_error



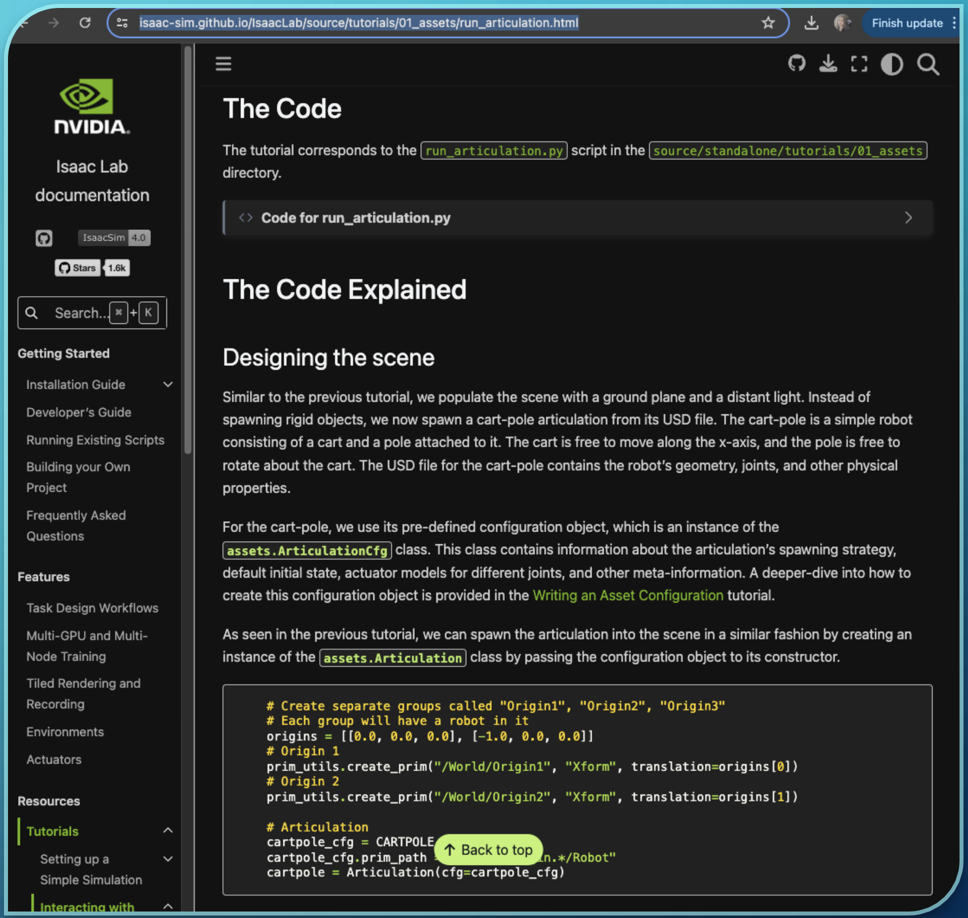
https://medium.com/@svm161265/when-and-why-to-use-p-pi-pd-and-pid-controller-73729a708bb5



https://electronics.stackexchange.com/questions/116749/control-theory-basics

ISAAC-LAB

- Articulation → robot
- Base code
- Python


isaac-sim.github.io/IsaacLab/source/tutorials/01_assets/run_articulation.html

NVIDIA

Isaac Lab documentation

Search...

Getting Started

- Installation Guide
- Developer's Guide
- Running Existing Scripts
- Building your Own Project
- Frequently Asked Questions

Features

- Task Design Workflows
- Multi-GPU and Multi-Node Training
- Tiled Rendering and Recording
- Environments
- Actuators

Resources

- Tutorials
  - Setting up a Simple Simulation

The Code

The tutorial corresponds to the `run_articulation.py` script in the `source/standalone/tutorials/01_assets` directory.

Code for run_articulation.py

The Code Explained

Designing the scene

Similar to the previous tutorial, we populate the scene with a ground plane and a distant light. Instead of spawning rigid objects, we now spawn a cart-pole articulation from its USD file. The cart-pole is a simple robot consisting of a cart and a pole attached to it. The cart is free to move along the x-axis, and the pole is free to rotate about the cart. The USD file for the cart-pole contains the robot's geometry, joints, and other physical properties.

For the cart-pole, we use its pre-defined configuration object, which is an instance of the `assets.ArticulationCfg` class. This class contains information about the articulation's spawning strategy, default initial state, actuator models for different joints, and other meta-information. A deeper-dive into how to create this configuration object is provided in the Writing an Asset Configuration tutorial.

As seen in the previous tutorial, we can spawn the articulation into the scene in a similar fashion by creating an instance of the `assets.Articulation` class by passing the configuration object to its constructor.

```
# Create separate groups called "Origin1", "Origin2", "Origin3"
# Each group will have a robot in it
origins = [[0.0, 0.0, 0.0], [-1.0, 0.0, 0.0]]
# Origin 1
prim_utils.create_prim("/World/Origin1", "Xform", translation=origins[0])
# Origin 2
prim_utils.create_prim("/World/Origin2", "Xform", translation=origins[1])

# Articulation
cartpole_cfg = CARTPOLE
cartpole_cfg.prim_path
cartpole = Articulation(cfg=cartpole_cfg)
```

https://isaac-sim.github.io/IsaacLab/source/tutorials/01_assets/run_articulation.html

METHODS

- Create articulation class for rigged gripper
- Apply ideal PD actuator to articulation
- Use “run_articulation.py” as base code to control gripper
    - Initialize gripper as open
    - While simulation is running, apply actuator and update step
    - If the sim step < 100: close gripper
    - If sim step >= to 100: open gripper
    - If sim step = 200: reset gripper
- Use “arms.py” as base code to control KG3
    - Initialize KG3 as upright with Robotiq attached
    - While simulation is running, apply actuator and update step
    - If sim step < 200: arm in rest position, gripper open
    - If sim step > 200: arm moves, gripper closes
    - If sim step = 400: reset assembled robot
- Troubleshooting: import pdb, IsaacLab docs/forums, ERROR messages, A.I., rubber ducky method

https://isaac-sim.github.io/IsaacLab/source/setup/sample.html

RESULTS

PhysX hack

```python
# resolve articulation root prim back into regex expression
root_prim_path = root_prims[0].GetPath().pathString
root_prim_path_expr = self.cfg.prim_path + root_prim_path[len(template_prim_path) :]
# -- articulation
self._root_physx_view = self._physics_sim_view.create_articulation_view(root_prim_path_expr.replace(".*", "*"))
# -- link views
# note: we use the root view to get the body names, but we use the body view to get the
#       actual data. This is mainly needed to apply external forces to the bodies.
physx_body_names = self.root_physx_view.shared_metatype.link_names
body_names_regex = r"(" + "|".join(physx_body_names) + r")"
# body_names_regex = f"{self.cfg.prim_path}/{body_names_regex}" # this is the original code... it doesn't account for any extra prims between "robot" and the link names
body_names_regex = f"{root_prim_path_expr}/{body_names_regex}" # it is essential to use the "root_prim_path_expr" when defining robot links/parts. root view and body view must be defined the same.
self._body_physx_view = self._physics_sim_view.create_rigid_body_view(body_names_regex.replace(".*", "*"))

# create ordering from articulation view to body view for body names
# note: we need to do this since the body view is not ordered in the same way as the articulation view
# -- root view
root_view_body_names = self.body_names
print("------root_view_body_names-----")
print(root_view_body_names)
# -- body view
prim_paths = self._body_physx_view.prim_paths[: self.num_bodies]
body_view_body_names = [path.split("/")[-1] for path in prim_paths]
# -- mapping from articulation view to body view
self._body_view_ordering = [body_view_body_names.index(name) for name in root_view_body_names]
self._body_view_ordering = torch.tensor(self._body_view_ordering, dtype=torch.long, device=self.device)
```

pdb and robot_assembler hack

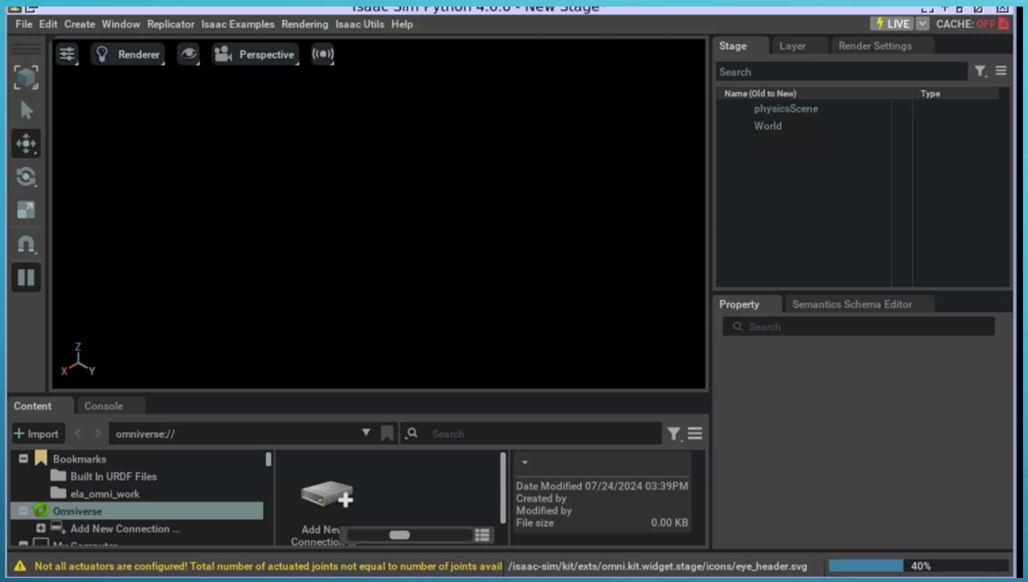
```
-> assembled_robot = robot_assembler.assemble_articulations(
(Pdb) c
> /isaac-sim/exts/omni.isaac.robot_assembler/omni/isaac/robot_assembler/robot_assembler.py(555)_move_obj_b_to_local_pos()
-> a_trans, a_orient = XFormPrim(base_mount_path).get_world_pose()
(Pdb) n
> /isaac-sim/exts/omni.isaac.robot_assembler/omni/isaac/robot_assembler/robot_assembler.py(557)_move_obj_b_to_local_pos()
-> a_rot = quats_to_rot_matrices(a_orient)
(Pdb) s
--Call--
> /isaac-sim/exts/omni.isaac.core/omni/isaac/core/utils/numpy/rotations.py(107)quats_to_rot_matrices()
-> def quats_to_rot_matrices(quaternions: np.ndarray, device=None) -> np.ndarray:
(Pdb) n
> /isaac-sim/exts/omni.isaac.core/omni/isaac/core/utils/numpy/rotations.py(116)quats_to_rot_matrices()
-> if len(quaternions.shape) == 1:
(Pdb)
```

```python
@staticmethod
def _move_obj_b_to_local_pos(base_mount_path, attach_path, attach_mount_path, rel_offset, rel_orient):
    # Get the position of base_mount_path as `a`
    import pdb
    pdb.set_trace()
    a_trans, a_orient = XFormPrim(base_mount_path).get_world_pose()

    a_trans = np.asarray(a_trans.cpu())
    a_orient = np.asarray(a_orient.cpu())

    a_rot = quats_to_rot_matrices(a_orient)
    rel_rot = quats_to_rot_matrices(rel_orient)
```

```python
# The attach_mount_path local xform is a free variable, and setting its world pose doesn't
# change the world pose of every part of the Articulation.

# We need to set the position of attach_path such that attach_mount_path ends up in the
# desired location.  Let the attach path location be `b`.

# t_bc denotes the translation that brings b to c
# r_bc rotates from b to c

t_bc, q_bc = XFormPrim(attach_mount_path).get_local_pose()

t_bc = np.asarray(t_bc.cpu())
q_bc = np.asarray(q_bc.cpu())

r_bc = quats_to_rot_matrices(q_bc)

b_rot = c_rot @ r_bc.T
b_trans = c_trans - b_rot @ t_bc
b_orient = rot_matrices_to_quats(b_rot)

XFormPrim(attach_path).set_world_pose(b_trans, b_orient)
```

RESULTS

Robot assembler



CONCLUSIONS

- USD files manipulated in IsaacLab must contain ONLY robot w/ articulation root
- PhysX error: To grab “body view” in articulation.py we must grab all prims in “root view”
  - Utility: run_articulation_robotiq.py
- Robot_assembler data type error: requires a numpy array in _move_obj_b_to_local_pos function but grabs a cuda tensor
  - Utility: kg3_test.py
- Velocity control allows grasp
- Changing gripper’s direction of motion using controller results in massive torque

FUTURE DIRECTIONS

- Train gripper in sim
- Follow target example
- Train hardware using sim data
- Deformable force sensors

https://medium.com/@sthanikamsanthosh1994/imitation-learning-behavioral-cloning-using-pytorch-d5013404a9e5
https://developer.nvidia.com/blog/closing-the-sim-to-real-gap-training-spot-quadruped-locomotion-with-nvidia-isaac-lab/
https://docs.omniverse.nvidia.com/isaacsim/latest/advanced_tutorials/tutorial_advanced_adding_new_manipulator.html