Rigging, controlling, and assembling robots in Nvidia Isaac-Sim Omniverse for imitation learning use

Ela Nuñez
Florida International University
Mentors: Roxana Leontie, PhD; Ken Stewart, PhD
(Summer 2024 – NRL, Washington DC)

Abstract: Imitation learning is a technique for training agents (robots) to perform tasks by imitating demonstrations from an expert. Gathering extensive expert datasets by manual teleoperation of robots can be both time-consuming and labor-intensive. A potentially more efficient approach involves creating a digital twin of the environment and the robot and collecting most data from simulation [1]. This report addresses a fundamental functionality needed for such a data collection: enabling grasping of simulated hardware available at NRL in Nvidia's Isaac-Sim Omniverse robotic simulator [2]. This report covers the technical challenges of adding and controlling a robotic manipulator, the Robotiq 2F85 gripper, and attaching it to a 7 degree-of-freedom (7 DOF) arm, the Kinova Gen3 arm, to be used for future expert data collection on Navy relevant tasks.

Introduction:

The motivation for this work was to simulate existing NRL hardware (robotic gripper and arm) in Nvidia Omniverse Isaac Sim, a real-time 3D graphics platform and physics simulator, and generate data that could potentially enable robots to perform Navy relevant tasks using imitation learning. Imitation learning (IL) traditionally uses data collected by a human expert teleoperating the robot through a complex task to create "policies" (a term referring to trained behaviors) which enable robots to accomplish the task similarly to the expert. This is advantageous for tasks that are either inaccessible or unsafe for humans but demand high success rates and reproducibility. However, collecting data with teleoperation is time consuming and expensive; therefore, we propose to collect data of tasks in simulation using Nvidia Isaac Sim.

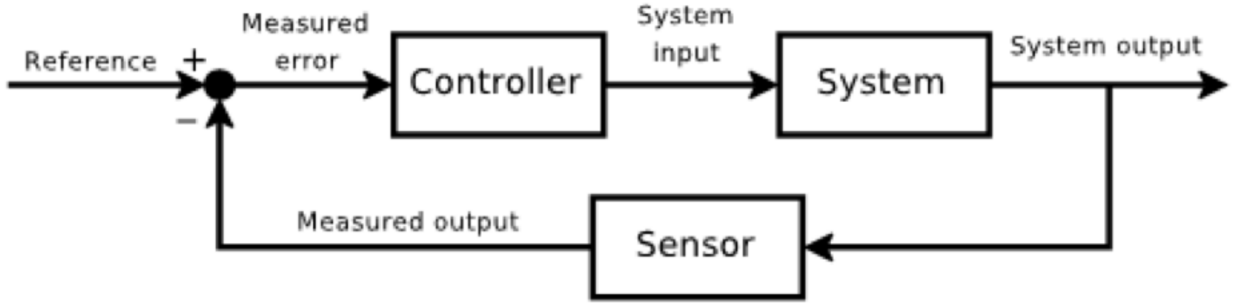
Some simple manipulation tasks are already documented in Isaac Lab, Nvidia's robot learning framework, among which include "close gripper," "follow target," "pick-place," and random motion generators such as "run articulation" [7, 8]. These scripts utilize articulated grippers to perform tasks suited for an arm/gripper imitation learning framework. The articulated gripper is designed with joints that enable them to open and close in simulation much like they would in physical hardware. Task datasets simulated in Omniverse can be gathered using Robomimic's framework and could then be used to actuate real hardware through a ROS2 interface [4, 9, 6]. However, it's necessary to demonstrate that NRL robotic hardware can be used to accomplish these and other Navy relevant tasks, thus presenting the problem of how to rig, control, and assemble our robots using Nvidia's framework for robot manipulation.

First, I investigated the Robotiq 2F85 2-finger gripper as it was one of the grippers of interest to model and control. Although there was an existing Nvidia Omniverse model, it's rigging was broken in Isaac Sim. I worked on fixing it and explored multiple control strategies, including position and velocity control. Robotic control strategies such as position and velocity are chosen based on the specific requirements of the manipulation task. Position control is typically used when precise placement of an object is crucial, such as in assembly tasks or delicate operations. Velocity control is employed when the speed of movement needs to be regulated, which is important for tasks requiring smooth and coordinated motion, like in pick-and-place operations. Each control method ensures that the robot performs the task effectively while accommodating the complexities of the environment and the nature of manipulation.

Next, to perform meaningful manipulation tasks, the gripper needed to be attached to a robotic arm: the 7 DOF Kinova Gen3 (KG3) robotic arm was used for this purpose. 7 DOF arms are particularly advantageous in environments with obstacles because they exhibit redundancy. Like the human arm, these robotic arms can achieve the same end effector (gripper) position though various arm configurations, allowing for greater flexibility and maneuverability around obstacles [10]. Since the KG3 and the Robotiq gripper are two separate model files, an assembly in Omniverse was needed to achieve our desired robot. Isaac-Sim's Robot Assembler allows us to do this by joining independent articulations and actuating them separately or at the same time [11]. Once assembled, the gripper and robotic arm become a single articulation tree – a "tree structure of rigid-body links and the joints between them" [17]. This assembled robot could then be controlled to generate data on Navy relevant complex tasks.

Properly configuring this articulation tree ensures that the robots can be controlled as intended. The KG3's articulation tree begins at its base link on the ground plane and extends upward as links are connected by joints [17]. The Robotiq gripper's articulation tree also begins at its base link, then branches out into 2 fingers which form separate, cyclic articulation loops. Managing and controlling cyclic kinematic trees can be more challenging than linear chains due to the need to account for the interactions within the cycles. These loops had to be broken to allow for a more traditional and simple ways to control the gripper using linear kinematic chains. Once the cyclic kinematic chains were broken, classical control theory could be applied to open and close the gripper.

Isaac Sim's control framework, called Isaac Lab, uses closed loop feedback systems to simulate robots. Closed loop systems are widely utilized in robotics because they enable agents to automatically correct errors and adjust their movements to achieve a desired position [23]. Roboticists create controllers to implement in a closed control loop, such as the one seen below [22]:

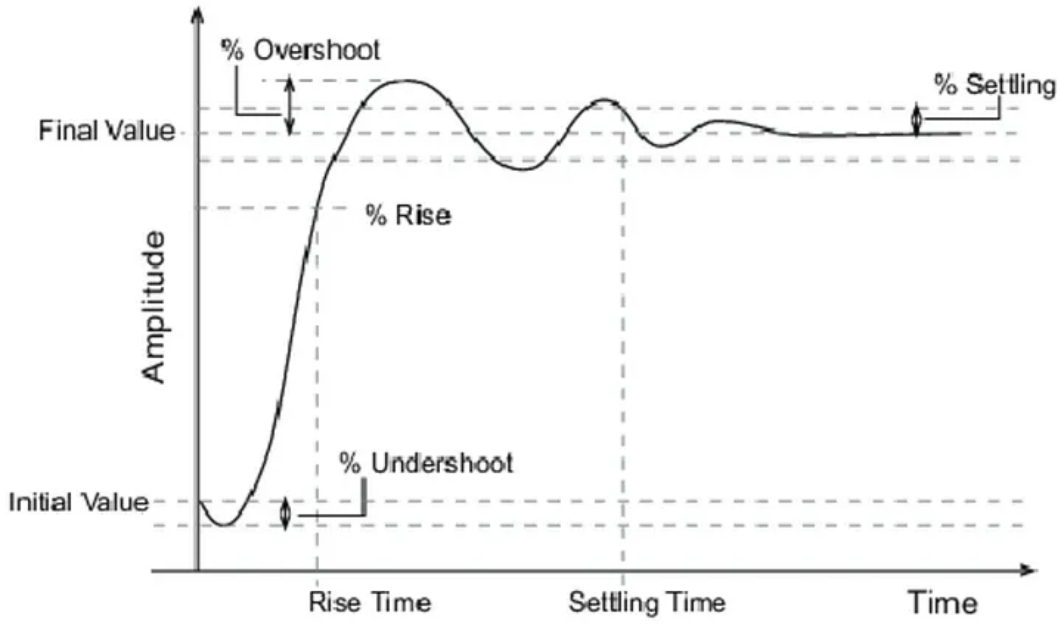


In a closed loop control: the sensor is the simulated/measured position, velocity, and torque data, and the controller is Proportional Derivative (PD) controller. As the sensor feeds updated measurements to the controller, the controller mathematically corrects error between current and desired measurements and gives its output back to the system instantaneously. A PD controller's classical equation – commonly used in robotic control theory – can be seen below:

effort = Kp * pos_error + Kd * vel_error

The control effort is returned to the robot, Kp is the proportional gain, pos_error is the positional error, Kd is the derivative gain, and vel_error is the velocity error (i.e. the derivative of the position error).

As a robot searches for the correct position, there is an oscillatory period that must be settled. Derivative gain settles oscillation of the target position until it is at its steady state (i.e. desired position), while proportional gain ensures that the robot reacts quickly after being given a new command by the controller. By tuning Kp and Kd, one can adjust the responsiveness and stability of the control system. An example of this stabilization can be seen below [21]:

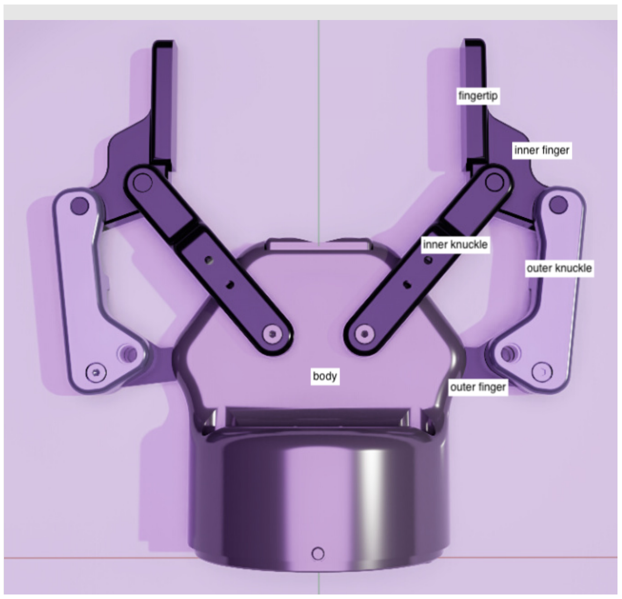


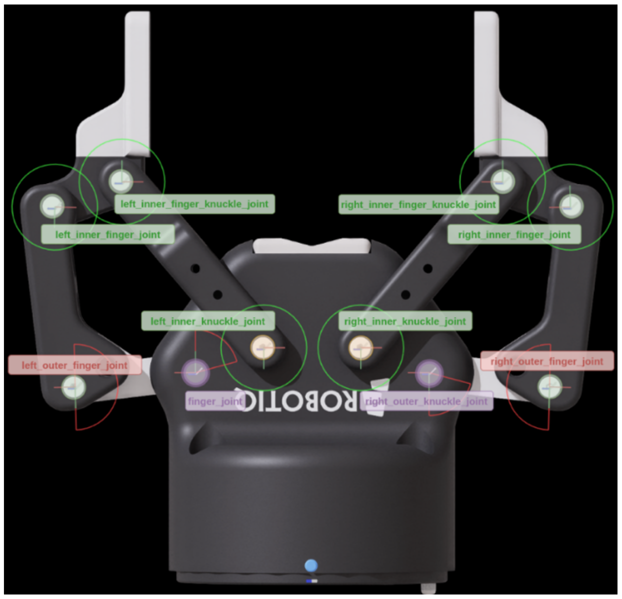
Materials and Methods:

Rigging a robot from scratch: The first task was to rig a Robotiq 2F85 CAD file downloaded from Robotiq's official website by following the "Rigging Complex Structures" tutorial in Isaac-Sim's documentation [13, 2]. The Robotiq could be downloaded as a .STEP file in its open or closed configuration: this work used the open configuration. It was then imported into SolidWorks to ensure that the origin of each part was in the center of the gripper's base.

After enabling the CAD converter as an extension in the Isaac-Sim GUI, the SolidWorks file was imported into the scene [14]. By selecting the robot in `Stage` and pressing F on the keyboard one can zoom focus on the selected prim. For the gripper fingertips to stop when closed, or around an object in simulation, collision must be enabled. To do so the file was first saved in a different format `File → Save flattened as…` (which also allows prim names to be modified on an imported file), and then robot collision meshes were selected by `[select] "Robotiq_2F85" → [check]`, and then `"Self-collision enabled"` was checked in the Articulation Root options. An important next step is to set the pivot point of the model to (0, 0, 0) in the (x, y, z) plane through `Tools → Pivot → Add pivot → Set pivot position to…`.

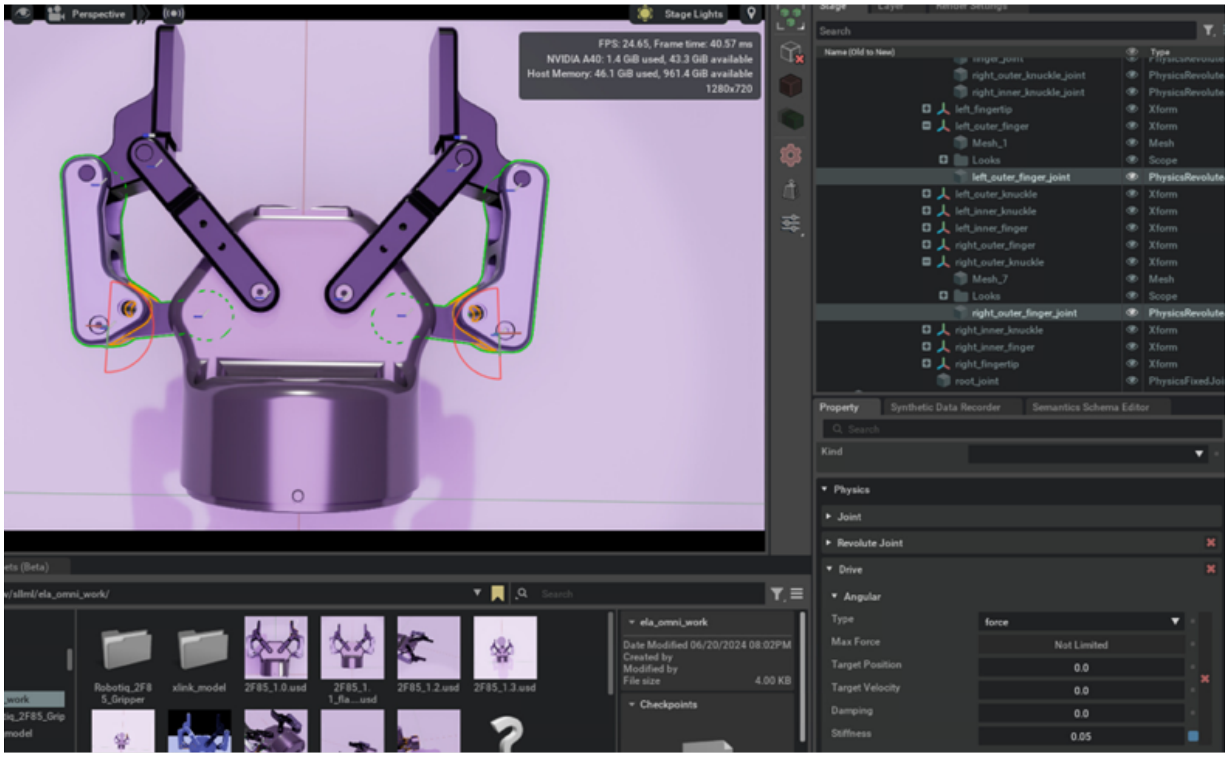
Additionally, we need to make sure the mesh of each link has the same pivot point as its prim to prevent malfunction while the simulation is running.





Next, joints for the 2F85 gripper were created and named based on the images/documentation above. Several joints had to have limits set. The "[left/right]_outer_finger_joint" require limits [0-180] degrees, while the "finger_joint" and "right_outer_knuckle_joint" require limits [0-75] degrees. Create joints by `[select] Body 1 → [ctrl select] Body 2 → Create → Physics → Joints → Revolute Joint → [change axis of joint to Z] → [set Local Positions 0 and 1 to 0.0 in Z (this will set all of the joints level to the X-Y plane)] → [rename joint]` to articulate the gripper. Work up the articulation tree during this process (i.e. for the "finger_joint", body 0 is the "base" and body 1 is the "left_outer_finger"; for the "left_outer_finger_joint" body 0 is the "left_outer_finger" and body 1 is the "left_outer_knuckle"). Body 1 will be the link whose axis the joint inherits [19]. Ensure that each joint is set to the exact center position of a screw between two links using `Perspective → [Top, Front, Right]` and set joint local positions [0, 1] to 0 in Z plane. Attach "[left/right] fingertip" to "[left/right]_inner_finger" with a fixed joint. To actuate the Robotiq's rigged CAD file in Omniverse, joints must have the same unique orientations in 3-space as pictured above. Graphically those directions use red for the X axis and green for the Y axis. By default, the model presents a cyclic tree for each of the fingers. These loops had to be broken to allow for a more traditional and simple way to control the gripper using linear kinematic chains.

To break articulation loops that exist in each finger we `[select] "[left/right]_inner_knuckle_joint" → Properties → Physics → Joint → [check] "Exclude from Articulation"`. In any articulation loop, the ideal joint to exclude has no limits, no resistance, and no drive [2]. To test joint functionality, add a fixed joint to the gripper's base (remove after), play the simulation, select `shift` and drag the fingertips with the mouse.


Perspective
Stage Lights
FPS: 24.65, Frame time: 40.57 ms
NVIDIA A40: 1.4 GiB used, 43.3 GiB available
Host Memory: 46.1 GiB used, 961.4 GiB available
1280x720
Robotiq_2F85_Gripper
xlink_model
2F85_1.0.usd
2F85_1.3.usd
ela_omni_work
Checkpoints

Name (Old to New)	Type
finger_joint	PhysicsRevolute
right_outer_knuckle_joint	PhysicsRevolute
right_inner_knuckle_joint	PhysicsRevolute
left_fingertip	Xform
left_outer_finger	Xform
Mesh_1	Mesh
Looks	Scope
left_outer_finger_joint	PhysicsRevolute
left_outer_knuckle	Xform
left_inner_knuckle	Xform
left_inner_finger	Xform
right_outer_finger	Xform
right_outer_knuckle	Xform
Mesh_7	Mesh
Looks	Scope
right_outer_finger_joint	PhysicsRevolute
right_inner_knuckle	Xform
right_inner_finger	Xform
right_fingertip	Xform
root_joint	PhysicsFixedJoint

Physics
Joint
Revolute Joint
Drive
Angular

Type	force
Max Force	Not Limited
Target Position	0.0
Target Velocity	0.0
Damping	0.0
Stiffness	0.05

Subsequently, we add angular joint drives to every revolute joint. This is done through `[ctrl, select]` *`all revolute joints`* `→ Properties → Add → Physics → Angular Drive`. Identify which joint or joints drive your robot's articulation. In this case, the fingers are driven by "finger_joint" and "right_outer_knuckle_joint". Following the "Rigging Complex Structures: Adding Joint Drives" section, give the "finger_joint" velocity control properties by setting stiffness to 0 in the joint drive. Velocity control allows the gripper to grasp using force. When setting joint drives from scratch, use the "Tuning Joint Drive Gains: Position Drive and Velocity Drive" sections to properly set your control method [18]. To give the fingers parallel motion, set "[left/right]_outer_finger_joint" stiffness to 0.05 as seen in the image above. There are additional joint properties to be added, which are detailed in the tutorial [2]. If possible, add a mimic joint to the "right_outer_knuckle_joint" with reference to the finger joint. In the "finger_joint" drive, set its target velocity to 80 to open the gripper and -80 to close the gripper while the simulation is running. Increase the time step to improve the accuracy of joints during simulation by clicking `Physics scene → Physics → Scene → Time steps per second = 300`. The robot may be rigged correctly but not have enough iterations to converge properly [12].

Control the Robotiq using Python: First, the articulation file `robotiq.py` was created to call Isaac-Sim's Robotiq 2F85 USD asset and set its properties. `Spawn` properties such as `solver_position_iteration_count`, `solver_velocity_iteration_count`, `sleep_threshold`, and `stabilization_threshold` had to be set to account for the USD's articulation root. The gripper's collision meshes were enabled as rigid bodies. Every revolute joint in the USD had a joint drive (excluding the "[left/right]_inner_knuckle_joint"), and therefore were initialized in the articulation file at their open position. Effort and velocity limits were set to 5.0 and 130 respectively to match the "Rigging Complex Structures" tutorial [2, 5]. PhysX uses radians/second while USD files use degrees/second; automation conversions should have been implemented for effort limits, velocity limits, stiffness, and damping [16]. For velocity control, damping should be set to something greater than 0, and stiffness should be 0 (except in the “[left/right]_outer_finger_joint” so that the fingers can move in parallel). For position control, stiffness should be adjusted to something greater than 0 until the joints converge, then damping may be added [18].

Tuning joints with position and velocity control required careful adjustments of damping and stiffness, one at a time, for optimal function of the controller. The stiffness and damping of each joint are accessed by the ideal PD actuator configuration and used for Kp and Kd respectively. A separate code was required to initialize the simulation, create an environment with the gripper, and send it commands through the PD actuator. A source code from Isaac Lab’s documentation, `run_articulation.py`, was modified to spawn a single gripper on one origin and orient it upright and called `run_articulation_robotiq.py` [8]. The rigged Robotiq file was used to measure joint positions in degrees when the fingers were both open and closed. These positions were converted to radians for coding purposes and stored as torch tensors in `run_articulation_robotiq.py` [16]. In position control, the target accounted for current joint positions, desired joint positions, and time step so that the gripper moved smoothly over time. During the first 200 time-steps of the simulation, the closed position tensor was the desired position, during the second 200 time-steps the open position tensor was the desired position, and every 400 times steps the simulation reset. For velocity control, every enabled joint was set to a target of 1.39626 rad/s (80 deg/s) to close the gripper, and -1.39626 rad/s to open it. With each time step, targets were sent to the simulation and corrected by the PD actuator so the gripper would converge to its desired position or velocity.

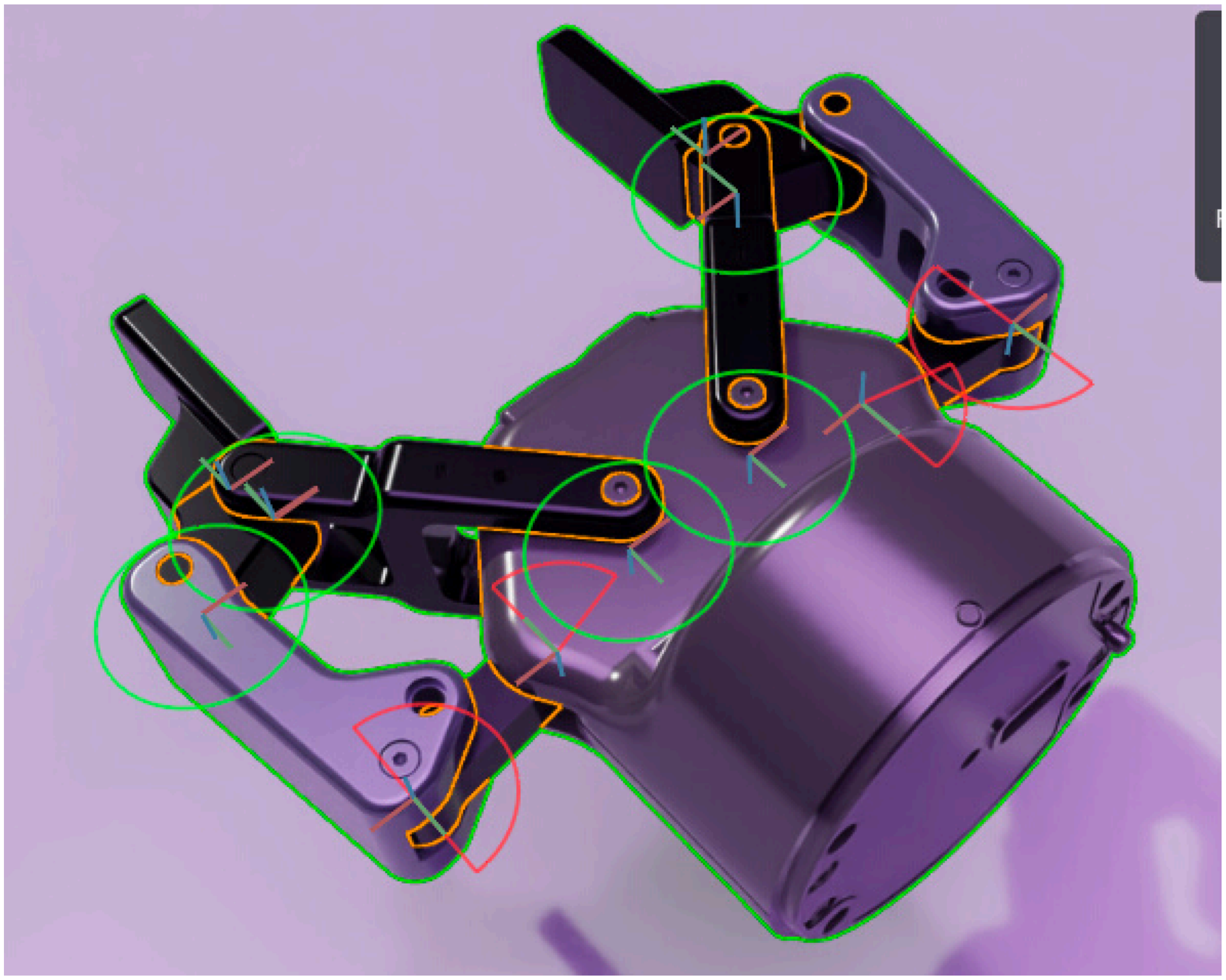
Robot Assembler: To attach the Robotiq gripper to a KG3 arm, a file was created using the base code `arms.py` [4]. First, the gripper and KG3 configs were imported into the file and all other robots were removed from the scene. Isaac Lab’s implicit actuator was imported to control the KG3 arm, and the ideal PD actuator was imported to control the gripper. The gripper used velocity control and the KG3 used position control. Both were articulations with PhysX applied. The robot assembler extension of Omniverse was enabled as follows [27]:

```
from omni.isaac.core.utils.extensions import enable_extension
enable_extension("omni.isaac.robot_assembler")
from omni.isaac.robot_assembler import RobotAssembler, AssembledRobot
```

To control the gripper and KG3 separately, set `single_robot` to be `false`, then return the gripper and KG3 as separate scene entities. To control them as a single robot with the arm as the articulation root, leave `single_robot` to be `true`. Given that the KG3 used position control and the gripper used velocity control, setting `single_robot` to be `false` would be best. However, in this implementation, `single_robot` was `true`.

Results:

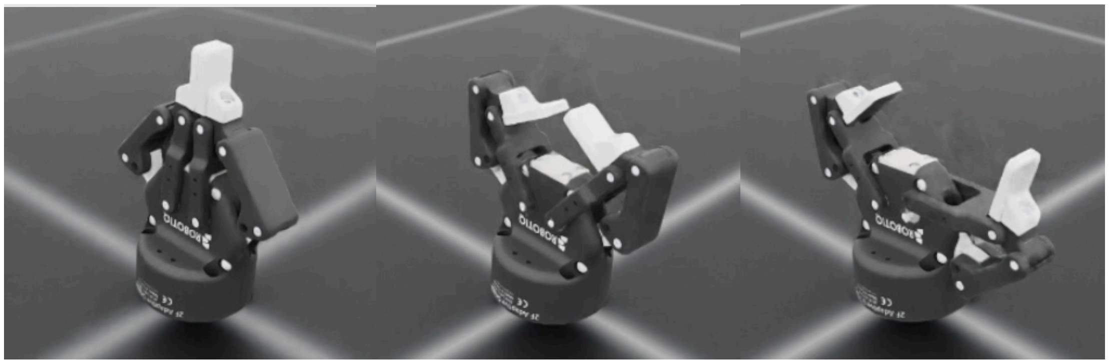
Rig a robot from scratch:



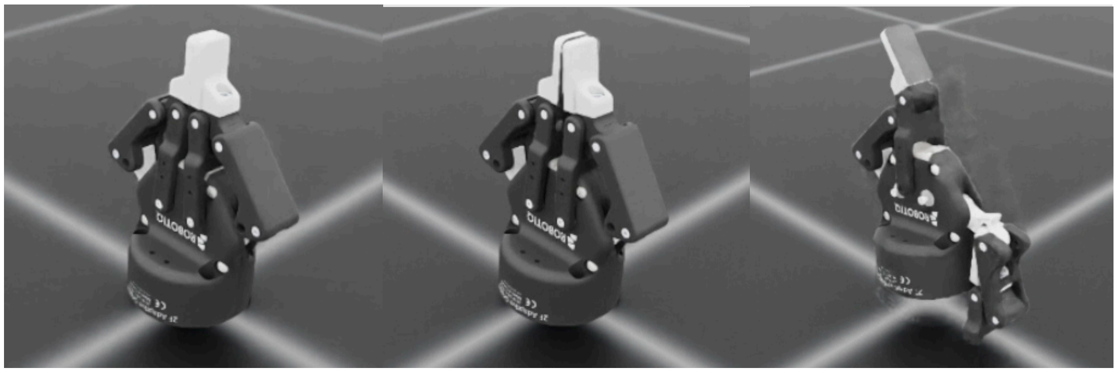
The mimic joint suggested in “Rigging Complex Structures” was unable to be added to the “right_outer_knuckle_joint” [2]. However, the gripper opened and closed as the “finger_joint” and “right_outer_knuckle_joint” target velocities were changed to 80 and -80 degrees/sec.

Control the robot using Python: When it was time to control the gripper in sim, I discovered that Nvidia's PhysX software had an error. In their original code (`articulation.py`), the gripper asset had an extra prim that wasn't accounted for when defining it as an articulation. After finding and fixing the problem, a bug report with the fix was submitted to Nvidia.

Next, using the code I developed, I was able to control the simulated robotic gripper to close fully using position and velocity control, but I was not as successful when trying to open it.

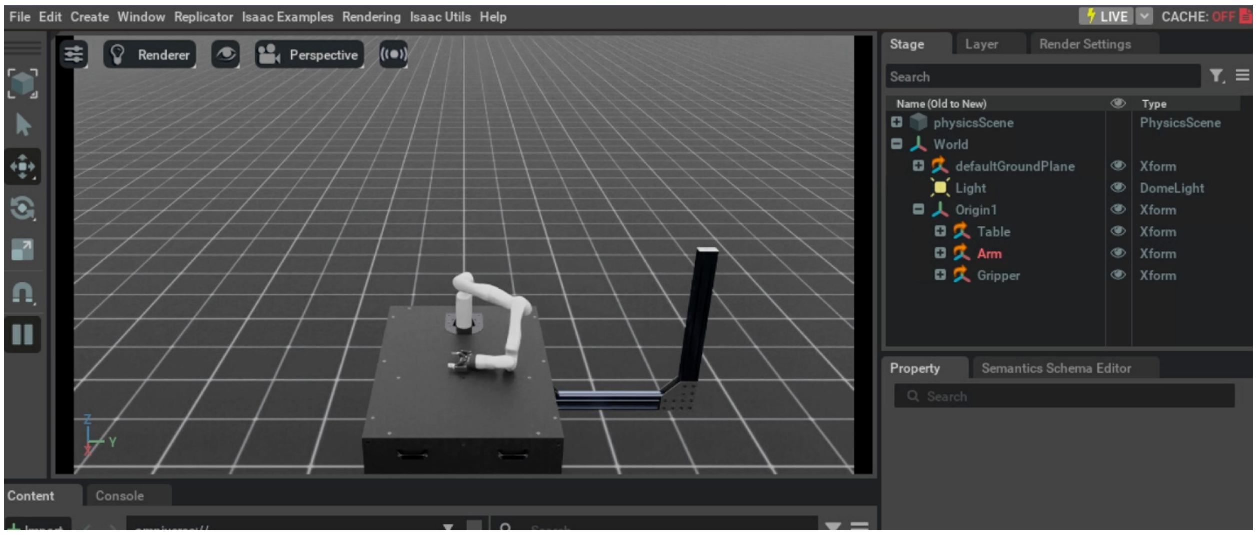


In the images above, the gripper was closed successfully but opened erratically using position control. When commanded to open, the fingertips did not remain parallel. The right finger hit the base and robot spun out of camera view. The symptom observed was that the joint positions increased beyond their USD limits, although torques remained the same throughout opening.

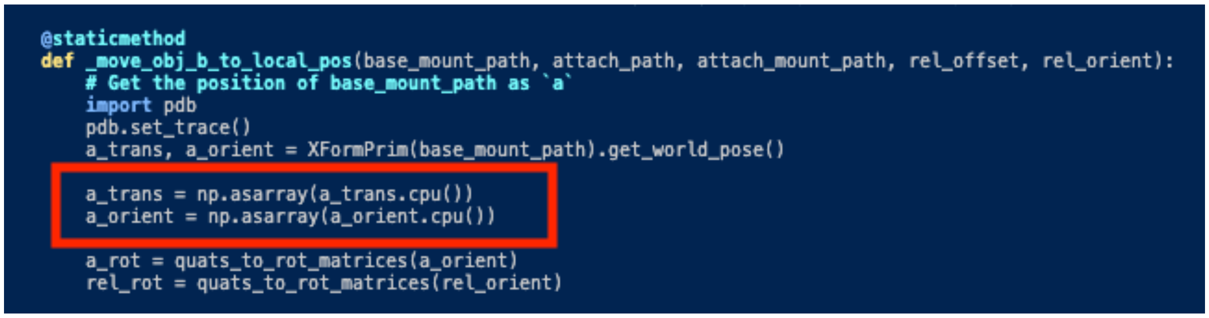


The images above show the gripper successfully closing but opening erratically using velocity control. During opening, the right finger reached massive velocities, overextended, and broke. When the gripper was commanded to open its joints' torques changed very inconsistently, from negative to positive, then back to negative. By the final image, all joints besides the "finger_joint" and "right_outer_knuckle_joint" had reached the maximum torque.

Robot Assembler: Being able to control the Robotiq griper was only part of the assignment. Next, I had to assemble them together.



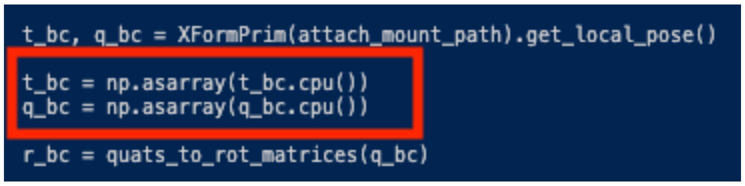
When implementing using the Robot Assembler extension I ran into another error that Nvidia didn't account for. I was able to trace the error to their `_move_obj_b_to_local_pos` function in the `robot_assembler.py` file, and then fix it using the data conversion that the code was expecting. A bug report and my fix were submitted to Nvidia to be incorporated in their next software release. My resolution involved changing the torch tensors to numpy arrays in two places of this "move object" function:



```
@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)
```



```
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)
```

Discussion and Conclusion:

I was able to demonstrate control of a rigged gripper in the Isaac Sim GUI, and, after fixing bugs in Nvidia's framework, partial automatic control of a gripper/arm assembly. The rigged gripper was successfully opened and closed using velocity and position commands in the Isaac Sim GUI. Two major bugs in released Nvidia software were identified and resolved to enable control of an assembled Robotiq gripper/KG3 articulation with a PD controller. However, more work is necessary to successively open the gripper after it was closed.

Despite the rigged grippers successful function in the GUI, there were a few problems with its expected functionality from the tutorial [2]. The gripper that was rigged from scratch did not allow addition of the mimic joint. This is possibly because rigging began in a version of Isaac-Sim that did not support mimic joints. To add a mimic joint, the user must have Isaac-Sim 4.0.0 or a more recent version. Nvidia's mimic joint might also require properties that I did not achieve by rigging from scratch. Nonetheless, the rigged gripper could be controlled by giving target inputs to both the "finger_joint" and "right_outer_knuckle_joint" simultaneously. It is possible to rig a gripper's CAD file in Isaac Sim by matching hardware joint properties, but there may be issues applying a mimic joint.

Once a sufficient level of control was demonstrated in the Isaac Sim GUI, the gripper could be closed automatically using Isaac Lab's framework. Both position and velocity control were implemented to explore how the gripper can be actuated to accomplish specific tasks. Velocity control ensures grasping of rigid objects while position control may not. This was found when the gripper was set up as following the tutorial's "Preparing For Tests" section and did not grasp a cylinder in position control [2]. When the gripper is given a position command, it reaches its target position but may not apply sufficient grasping force to the object. When the gripper is given a velocity command, it accelerates toward a target velocity and applies grasping force. Position control is more suited for operations requiring precise movement, such as assembly, and velocity control is best for tasks where smooth and speedy motions are beneficial, such as pick-and-place.

After closing the gripper, opening it did not work for position or velocity control. Position control changed the target position with each time step, which resulted in the joints going past what their limits should be. Giving limits to all joints in the gripper's articulation file, `robotiq.py`, would be beneficial. Velocity control generated too much torque, and the gripper's right finger broke quickly. Ideally, the gripper should be resting before given a torque command in the other direction. Otherwise, it likely overcompensates for the change in direction by doubling its torque command, but then has trouble attenuating this overcompensation. Enforcement of joint limits in joints that were unconstrained and further tuning of Kp and Kd (stiffness and damping) are needed to improve these control mechanisms.

Finally, I demonstrated that the Robot Assembler could be used to create a new, controllable robot by assembling the Robotiq gripper and KG3 in Isaac Sim. The robot assembler controls a gripper and arm as a single unit when `single_robot = true`, but to account for individual control of each articulation `single_robot` must be `false`. Then, the gripper can be given a target velocity to close or open, and the arm may be given a target position to move in 3-space. If there is a way to control both the gripper and arm with the same command for a specific task, it is likely by rigging the robot with a predefined controller from Omniverse's "Adding a

New Manipulator" tutorial [7]. Controlling the robots while accounting for their configuration differences could then be used to collect data for imitation learning tasks.

Future Directions:

This project demonstrated successful rigging of a gripper and tested control mechanisms of the gripper in Isaac Sim, but more work is needed to fully enable data collection using this solution. Ideas include adding another sensory measurement to the system for enhanced grasping capabilities and implementing another Isaac Sim extension for stable control.

By rigging a robot from its CAD file, users may replace parts of the model to enhance robot sensitivity. In this case, the gripper's fingertips could be replaced by force sensing fingertips that adjust grip through the controller. Dr. Leontie has acquired TacnIQ deformable force sensors, which would allow the Robotiq to have a sensitive grip on objects as fragile as eggs and deformable as sponges [24]. The reactive forces applied by the objects on the gripper and subsequent deformation of the gripper fingertips may be applied as additional sensory feedback to tasks in software according to [25]. To create the modified gripper in Isaac Sim, a CAD model of the force sensors would be obtained and connected by a fixed joint to the [left/right]_outer_finger. Rendering these deformable force sensors in software and hardware would be beneficial for precise, sensitive grasping of deformable objects during complex tasks.

Closing and opening the gripper consecutively in simulation may be accomplished by actuating the gripper as a Parallel Gripper manipulator instead of an articulation [26]. This manipulator only defines and activates 2 joints at the gripper root (base): "finger_joint" and "right_outer_knuckle_joint". It initiates an open position, a closed position, and "action_deltas" – the change in position to apply when opening or closing the gripper. The Parallel Gripper manipulator has been proven to work in "Adding a New Manipulator" but requires debugging as my attempt at using it resulted in errors that must first be addressed [7].

The Robotiq gripper and KG3 robot assembly may be used with a Pick Place task once the omni.isaac.manipulators import is working. However, first it may be best to apply the ideal PD actuator as the KG3's control model for consistency in the assembled robot. Currently the KG3 uses the implicit actuator while the gripper uses the ideal PD actuator, but Nvidia recommends not using the implicit actuator to demonstrate control in Isaac Sim. Once the task is successful in simulation, digital twins of the environment and robot will be created to collect datasets an order of magnitude faster than human teleoperation [1, 3].

References:

1. Garg, D. (2022, November 1). *Learning to imitate*. SAIL Blog. https://ai.stanford.edu/blog/learning-to-imitate/
2. *Rigging complex structures*. Omniverse IsaacSim latest documentation. (2024, July 28). https://docs.omniverse.nvidia.com/isaacsim/latest/advanced_tutorials/tutorial_advanced_rigging_complex_structures.html
3. *Data Logging*. Omniverse Robotics documentation. (2023, February 14). https://omniverse-content-production.s3-us-west-2.amazonaws.com/Assets/Isaac/Documentation/Isaac-Sim-Docs_2021.2.1/app_isaacsim/app_isaacsim/tutorial_advanced_data_logging.html
4. *Running existing scripts*. Isaac Lab documentation. (2024, July 26). https://isaac-sim.github.io/IsaacLab/source/setup/sample.html
5. Robotiq Inc. (2021). *Robotiq 2F-85 & 2F-140: Instruction Manual*. Robotiq Inc. Retrieved from https://assets.robotiq.com/website-assets/support_documents/document/2F-85_2F-140_General_PDF_20210623.pdf?_ga=2.92264026.1550868713.1719490799-2143217054.1717177701&_gac=1.187253466.1718982810.CjwKCAjwydSzBhBOEiwAj0XN4Jqv7VtKUmaBi6tAA_ICswIkg85GOOgbh-lAJ4sd3ULdPFvVo2GlPRoCibwQAvD_BwE.
6. *Ros 2 tutorials (linux & windows)*. Omniverse IsaacSim latest documentation. (2024b, July 28). https://docs.omniverse.nvidia.com/isaacsim/latest/ros2_tutorials/index.html
7. *Adding a new manipulator*. Omniverse IsaacSim latest documentation. (2024a, July 28). https://docs.omniverse.nvidia.com/isaacsim/latest/advanced_tutorials/tutorial_advanced_adding_new_manipulator.html
8. *Interacting with an articulation*. Isaac Lab documentation. (2024a, July 26). https://isaac-sim.github.io/IsaacLab/source/tutorials/01_assets/run_articulation.html
9. ARISE-Initiative. (n.d.). *Robomimic: A modular framework for robot learning from demonstration*. GitHub. https://github.com/ARISE-Initiative/robomimic
10. *What is the advantage having seven degrees of freedom in a robotic arm?*. Robotics Stack Exchange. (2020). https://robotics.stackexchange.com/questions/19732/what-is-the-advantage-having-seven-degrees-of-freedom-in-a-robotic-arm
11. *Assembling robots and rigid bodies*. Omniverse IsaacSim latest documentation. (2024b, July 28). https://docs.omniverse.nvidia.com/isaacsim/latest/advanced_tutorials/tutorial_advanced_assembling_robots.html
12. *Nvidia Omniverse Physics Extension - rigging a desk lamp part 4: Articulations and drives*. YouTube. (2022, August 16). https://www.youtube.com/watch?v=Orb_CEb9edc&t=6s
13. *2F-85 and 2F-140 grippers*. Robotiq. (n.d.). https://robotiq.com/products/2f85-140-adaptive-robot-gripper?ref=nav_product_new_button
14. *CAD converter*. Omniverse Extensions latest documentation. (2024, July 28). https://docs.omniverse.nvidia.com/extensions/latest/ext_cad-converter.html
15. NVIDIA Omniverse. (2022, August 16). *Nvidia Omniverse Physics Extension - rigging a desk lamp part 3: Rigging joints*. YouTube. https://www.youtube.com/watch?v=gm5y_XrgyUw

16. *Isaac Sim Conventions*. Omniverse IsaacSim latest documentation. (2024c, July 28). https://docs.omniverse.nvidia.com/isaacsim/latest/reference_conventions.html
17. *Articulations*. Omniverse Extensions latest documentation. (2024a, July 28). https://docs.omniverse.nvidia.com/extensions/latest/ext_physics/articulations.html
18. *Tuning joint drive gains*. Omniverse IsaacSim latest documentation. (2024f, July 28). https://docs.omniverse.nvidia.com/isaacsim/latest/advanced_tutorials/tutorial_advanced_joint_tuning.html
19. NVIDIA Omniverse. (2021, March 15). *Intro to physics part 2: Setting up joints in Nvidia omniverse USD composer*. YouTube. https://www.youtube.com/watch?v=YBmUYKrbLlw&t=284s
20. Isaac-Sim. (n.d.). *Isaac-SIM/IsaacLab: Unified framework for robot learning built on Nvidia Isaac Sim*. GitHub. https://github.com/isaac-sim/IsaacLab
21. VM, S. (2021, October 6). *When and why to use P, PI, PD and PID controller?*. Medium. https://medium.com/@svm161265/when-and-why-to-use-p-pi-pd-and-pid-controller-73729a708bb5
22. *Control theory basics*. Electrical Engineering Stack Exchange. (2013). https://electronics.stackexchange.com/questions/116749/control-theory-basics
23. *Control theory*. Control Theory - an overview | ScienceDirect Topics. (2023). https://www.sciencedirect.com/topics/social-sciences/control-theory
24. *Tacniq*. TACNIQ. (n.d.). https://www.tacniq.ai/
25. Narang, Y., Sundaralingam, B., Macklin (NVIDIA), M., Mousavian, A., & Fox, D. (2021, June 5). *Sim-to-real for robotic tactile sensing via physics-based simulation and learned latent projections*. Research. https://research.nvidia.com/publication/2021-06_sim-real-robotic-tactile-sensing-physics-based-simulation-and-learned-latent
26. *Manipulators*. Manipulators [omni.isaac.manipulators] - isaac_sim 4.1.0-rc.7 documentation. (2024, July 19). https://docs.omniverse.nvidia.com/py/isaacsim/source/extensions/omni.isaac.manipulators/docs/index.html?highlight=parallelgripper#omni.isaac.manipulators.grippers.ParallelGripper
27. *[question] using the Isaac Sim robotAssembler extension with the INTERACTIVESCENECFG class to attach two robots · issue #405 · Isaac-SIM/Isaaclab*. GitHub. (n.d.). https://github.com/isaac-sim/IsaacLab/issues/405