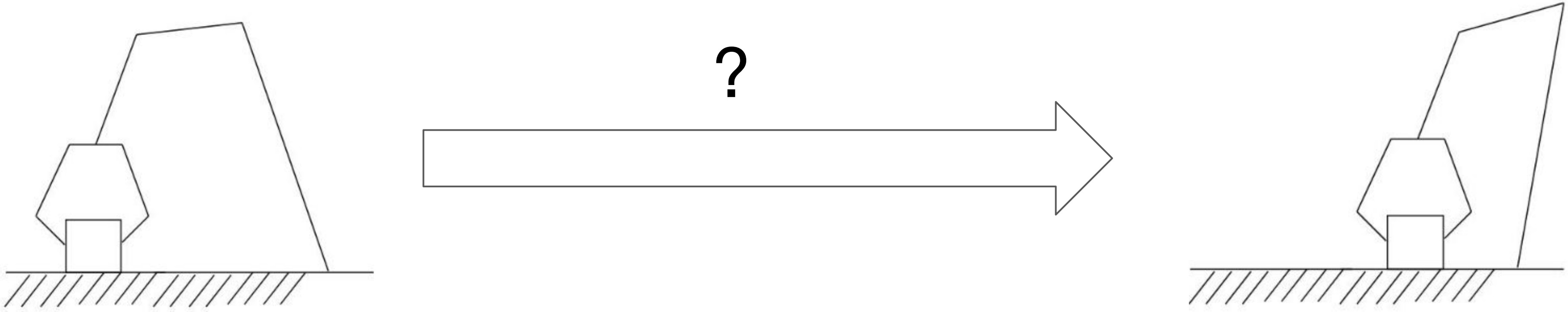
Outline

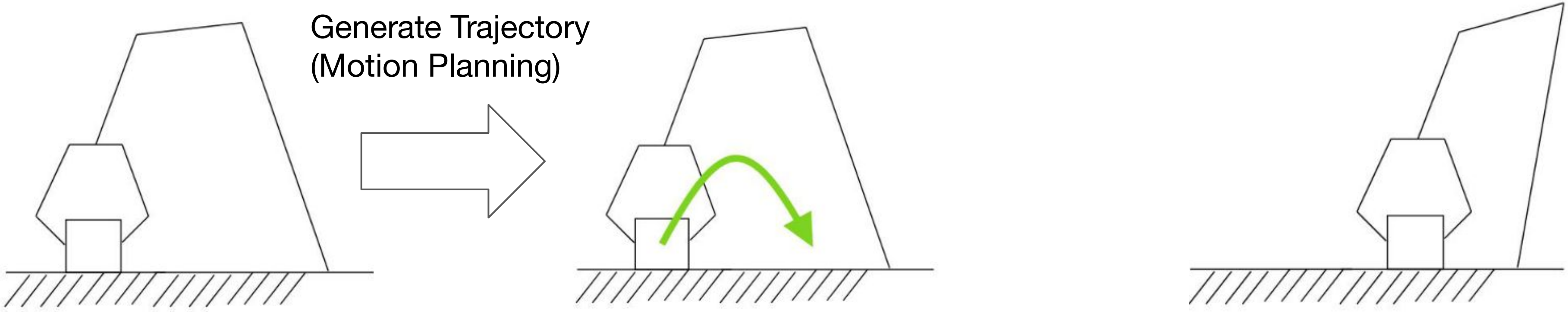
- Plan and Control
- Practices to Debug Simulators
  - Assets, physics, rendering, controller

Plan and Control



Plan and Control

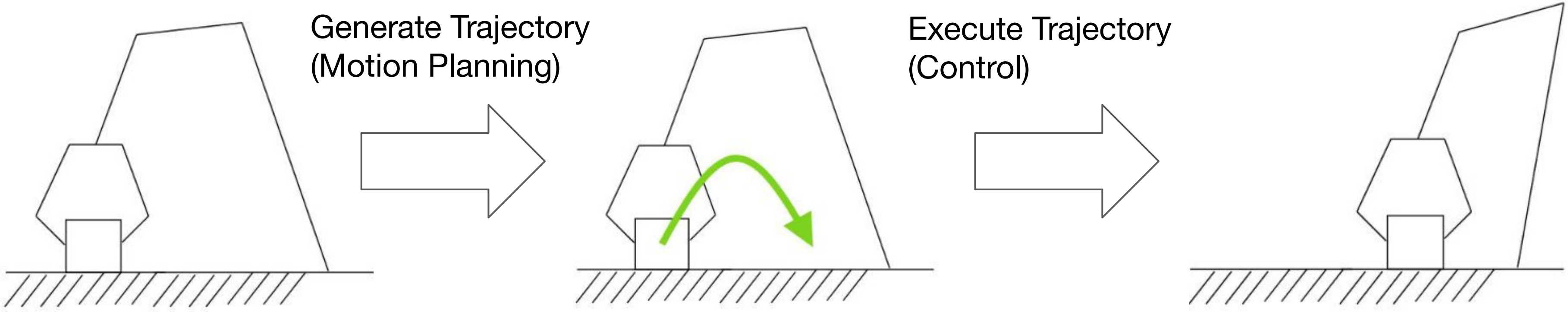
A popular pipeline in classic robotics is planning and control.



Motion planning generates a trajectory (position, velocity, and acceleration) of the robot.

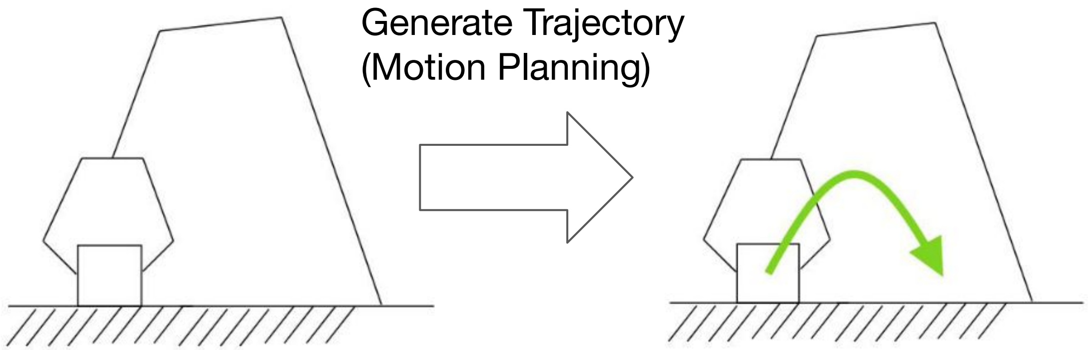
Plan and Control

A popular pipeline in classic robotics is planning and control.



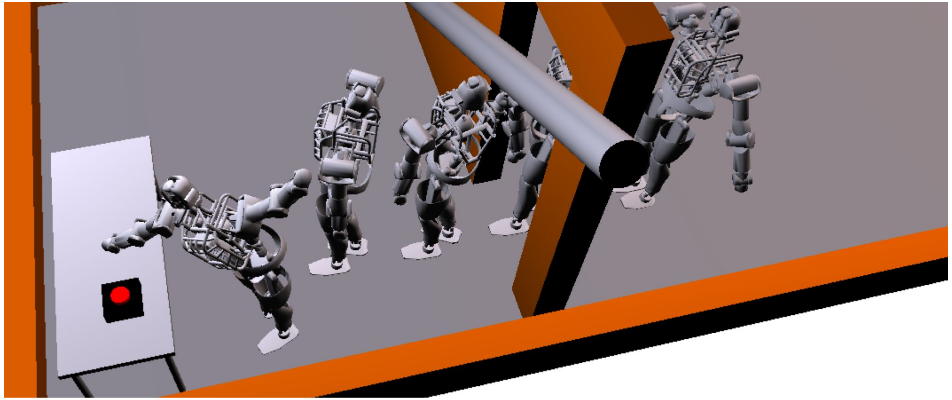
Motion planning generates a trajectory (position, velocity, and acceleration) of the robot.
Control executes the trajectory.

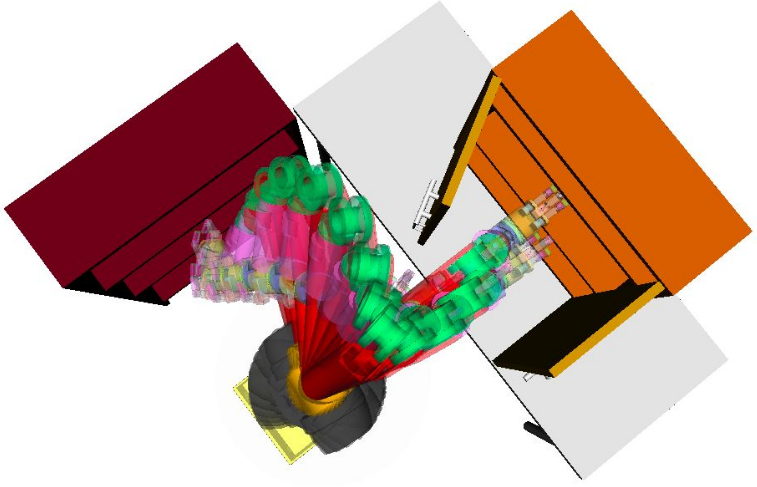
Motion Planning



Motion Planning

- Task: move a robot from one pose to another





Ratliff N, Zucker M, Bagnell J A, et al. CHOMP: Gradient optimization techniques for efficient motion planning, ICRA 2009
Schulman, John, et al. Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization, RSS 2013

Motion Planning

- Task: move a robot from one pose to another
- Assumptions
  - We know the start and goal pose
  - We can verify if a given pose is valid (usually means collision-free)
  - We can verify whether a pose is reachable from another pose using some simple control strategy

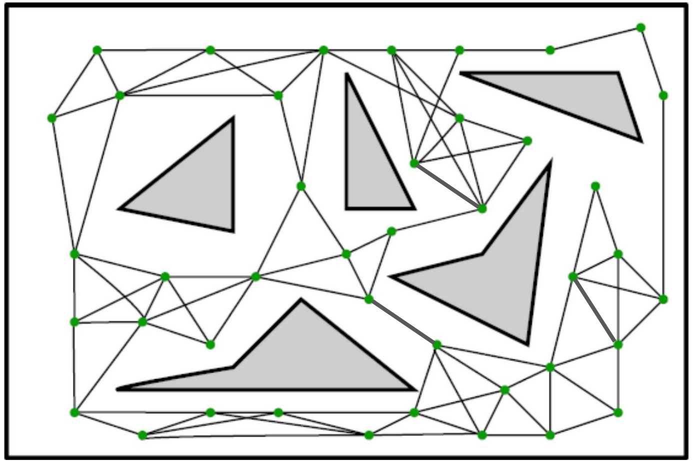
Motion Planning

- Task: move a robot from one pose to another
- Assumptions
  - We know the start and goal pose
  - We can verify if a given pose is valid (usually means collision-free)
  - We can verify whether a pose is reachable from another pose using some simple control strategy
- Algorithms
  - Rapidly-exploring random tree (RRT)
  - Probabilistic roadmap method (PRM)

Motion Planning Example: PRM

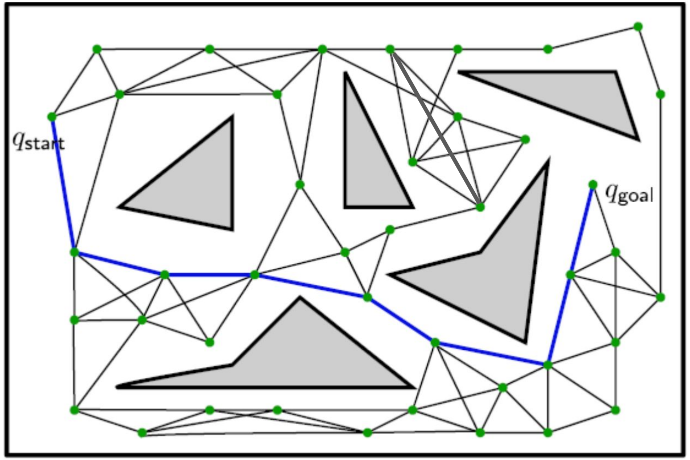
Motion Planning Example: PRM

- Phase 1: Map construction
  - Randomly sample collision-free configurations
  - Connect every sampled state to its neighbors
  - Connect the start and goal states to the graph



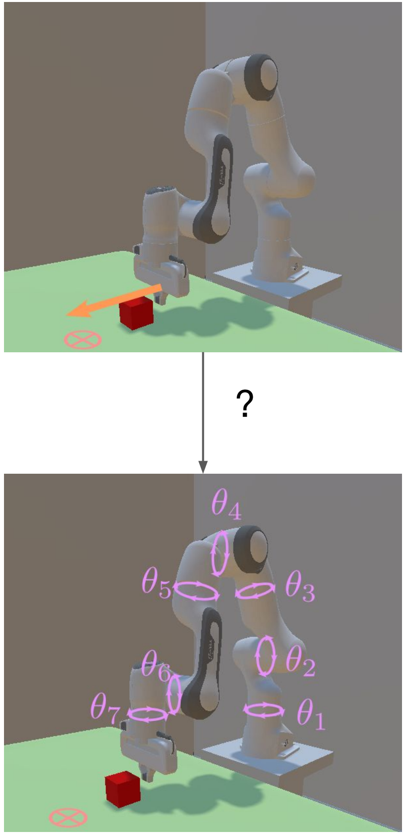
Motion Planning Example: PRM

- Phase 2: Query
  - Run path finding algorithms like Dijkstra



How to Find a Robot Pose For Grasping?

- Some tasks (such as grasping) require moving the gripper to a position.
- How do we find the robot pose of a given gripper pose?



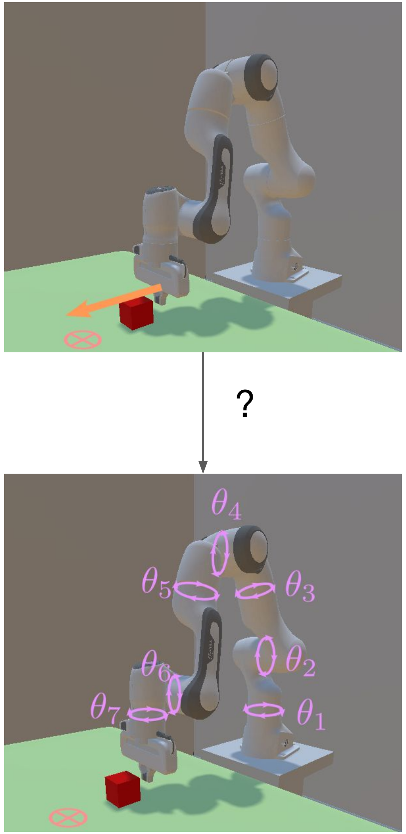
How to Find a Robot Pose For Grasping?

- Some tasks (such as grasping) require moving the gripper to a position.
- How do we find the robot pose of a given gripper pose?
  - **Inverse Kinematics (IK)**

```
robot_model = robot.create_pinocchio_model()

joint_positions, success, error = robot_model.compute_inverse_kinematics(
    link_idx,
    target_pose,
    active_qmask = joint_mask  # joints with mask value 1 are allowed to move
    max_iterations = 100
)
```

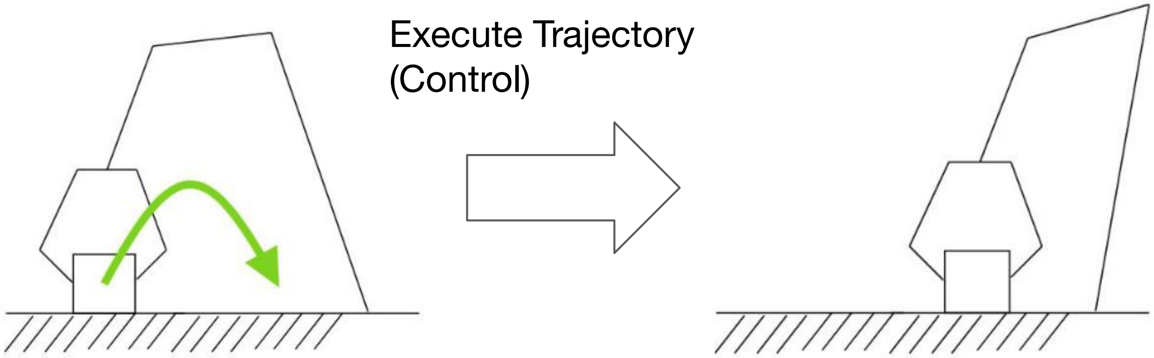
Code in SAPIEN



Time Parameterization

- PRM/RRT gives a path with discrete joint positions q_d
- A time parameterization algorithm converts the path q_d to a joint **trajectory** $(q_d, \dot{q}_d, \ddot{q}_d)$ with time.

Control



Control

- Robotic control executes a given trajectory $(q_d, \dot{q}_d, \ddot{q}_d)$ by controlling the joint torques τ
    - q represents the joint positions of a robot
- The problem is known as **Inverse Dynamics**

$$\tau = \mathrm{ID}(\ddot{q}; q, \dot{q})$$

Recall Last Lecture

Lagrangian Equation

- Lagrangian equation in vector form:

$$\tau = \boldsymbol{M}^b(\theta)\ddot{\theta} + C^b(\theta, \dot{\theta})\dot{\theta} + g^b(\theta)$$

  - $C^b_{ij}(\theta, \dot{\theta}) := \sum_{k=1}^{n} \Gamma^b_{ijk}\dot{\theta}_k$ is called the **Coriolis matrix**
    - Recall that in the body-frame Newton Euler equation, we also have a Coriolis term that comes from the derivate of rotational inertia. It was used to compensate for the rotational acceleration of the body frame
    - This $C^b_{ij}(\theta, \dot{\theta})$ also comes from taking the derivative of $\boldsymbol{M}^b$ w.r.t. θ. Because $\boldsymbol{M}^b$ and $\boldsymbol{\xi}^b$ are described in the body frame in our derivation, we also need this Coriolis term to compensate for the movement of the body frame.
  - $g^b(\theta)$ is due to gravity in our derivation. If there are other external forces (e.g., friction), it would also show up here.

Control

- Robotic control executes a given trajectory $(q_d, \dot{q}_d, \ddot{q}_d)$ by controlling the joint torques τ
    - q represents the joint positions of a robot

$$\tau = \mathrm{ID}(\ddot{q}; q, \dot{q}) = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q)$$

Inertia matrix

Coriolis matrix

Gravity & other forces

Control

- What we have
  - Trajectory $(q_d, \dot{q}_d, \ddot{q}_d)$
  - Inverse dynamics: $\tau = \mathrm{ID}(\ddot{q}; q, \dot{q})$
- Ideally, using τ computed from $\ddot{q}_d$ gives a perfect trajectory.
- However, the real world is not perfect. What if there is some error?

$$e = q - q_d$$

PD Control

- The PD control law has the form
  $\tau = -K_v \dot{e} - K_p e$ where $K_v, K_p \in \mathbb{S}^+$ $e = q - q_d$
- Intuitively
  - When the position lags behind ($e < 0$), increase τ to catch up
  - When it is moving too slow ($\dot{e} < 0$), also increase τ to catch up
  - Inverse dynamics is not used at all!

PD Control

- PD control has no convergence guarantee in general
- When it converges, often $e \neq 0$
- How to fix it?

- Combine PD control and inverse dynamics. (Augmented PD control)

$$\tau = \mathrm{ID}(\ddot{q}; q, \dot{q}) - K_v \dot{e} - K_p e$$

PID Control

- To mitigate steady-state errors, an integral term is often added.

PID: $$\tau = -K_v \dot{e} - K_p e - K_i \int_0^t e(t)dt$$

Augmented PID: $$\tau = \mathrm{ID}(\ddot{q}; q, \dot{q}) - K_v \dot{e} - K_p e - K_i \int_0^t e(t)dt$$

$$K_v, K_p, K_i \in \mathbb{S}^+ \qquad e = q - q_d$$

Example: PD Velocity Controller

- Velocity controller
  - Constant velocity trajectory; acceleration is 0
  - Do not care about position error; $K_p = 0$

$$\tau = \mathrm{ID}(0; q, \dot{q}) - K_v \dot{e}$$

```
for joint in robot.get_active_joints():
    # stiffness: diagonal of Kp
    # damping: diagonal of Kv
    joint.set_drive_property(stiffness=0, damping=10.0)

robot.set_drive_velocity_target(joint_velocity_target)  # set PD control velocity
passive_force = robot.compute_passive_force(gravity=True, coriolis_and_centrifugal=True)  # ID(0;q,q̇)
robot.set_qf(passive_force)  # augment PD control with ID
```

Code in SAPIEN

Use Control in MDP Modeling

- When an RL work says: *we use "velocity control" or "position control" as action*. What does that mean?

Use Control in MDP Modeling

- The action in an MDP can be “target joint velocity” or “target joint position” for a controller.

Use Control in MDP Modeling

- The action in an MDP can be “target joint velocity” or “target joint position” for a controller.
- A controller (such as PD) is used to convert this velocity or position signal to joint torques, which are then used to drive the robot.

Use Control in MDP Modeling

- The action in an MDP can be “target joint velocity” or “target joint position” for a controller.
- A controller (such as PD) is used to convert this velocity or position signal to joint torques, which are then used to drive the robot.
- Joint velocity/position may be a better choice for MDP action (than torque) due to learnability and sim-to-real transferability.

More About Control

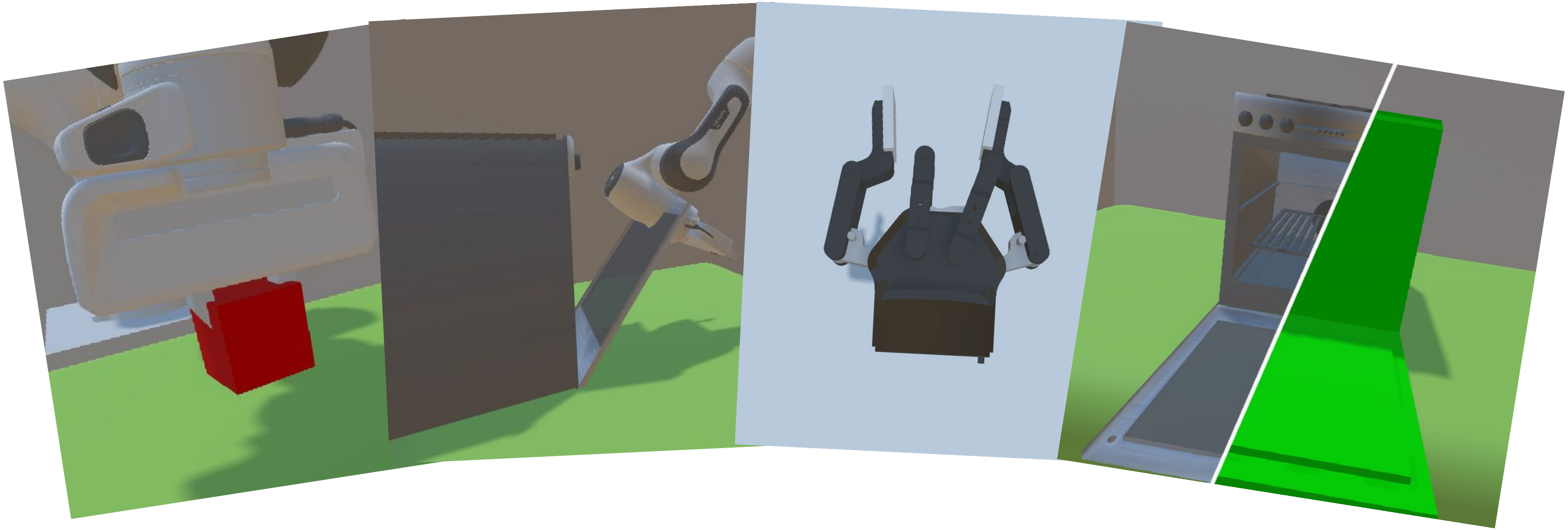
- Control focuses on stability and robustness
- A lot of literature
  - Optimal control
  - Feedforward/feedback control (including PD)
  - Robust control
  - Self-organized control
  - Stochastic control
  - …
- Optimal control has a strong connection with RL

Summary

- Classic robotics
  - Planning
    - RRT, PRM
    - Generates kinematic trajectory $(q_d, \dot{q}_d, \ddot{q}_d)$
  - Control
    - Inverse dynamics $\tau = \mathrm{ID}(\ddot{q}; q, \dot{q})$
    - Torque, PD, PID, Augmented PD
    - Find appropriate torque τ to follow trajectory $(q_d, \dot{q}_d, \ddot{q}_d)$

Practices to Debug Simulators

Simulations can Produce Many Unexpected Behavior



Overview

- We are going to talk about
  - How to identify potential problems when a simulation environment behaves unexpectedly.
  - How to debug and improve an environment.

Code used in this section https://github.com/haosulab/cvpr-tutorial-2022

Outline

- Causes of common bugs: conventions in robotics
- Causes of common bugs: simulation assets
- Causes of common bugs: physical solver
- Causes of common bugs: renderer
- Causes of common bugs: controller
- Environment speed

Outline

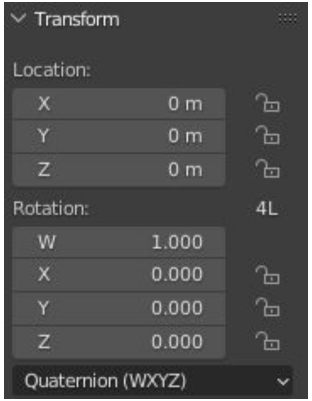
- Causes of common bugs: conventions in robotics
- Causes of common bugs: simulation assets
- Causes of common bugs: physical solver
- Causes of common bugs: renderer
- Causes of common bugs: controller
- Environment speed

Causes of common bugs: Conventions in Robotics

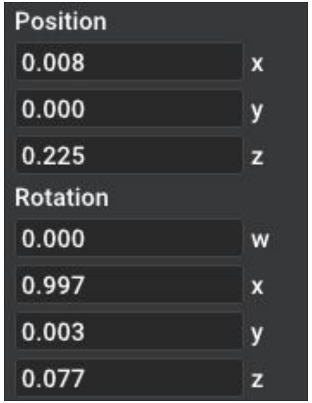
- Quaternion representations
- Euler-angle representations
- Default coordinate frames
- Joint order of different software and real robot

Quaternion Representations

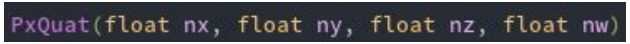
- Quaternion has 2 conventions:
  - XYZW (Vector First):
    - ROS, PyBullet, PhysX, scipy, Unity
  - WXYZ (Scalar First):
    - SAPIEN, transforms3d, Eigen, Blender, MuJoCo, V-Rep, PyTorch3d, numpy-quaternion
  - Everytime you use quaternion, check the convention.



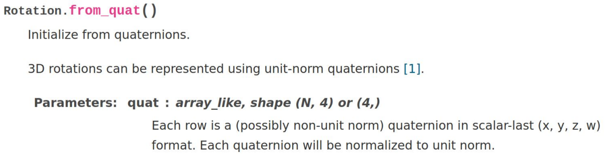
Blender



SAPIEN



PhysX



scipy

Euler Angle Representations

- Euler Angle has even more conventions
  - 24 conventions (includes Tait–Bryan angles)
- Even for an "xyz" convention, there are two possibilities:
  - Intrinsic rotations(**r**otating): coordinate axes attached to a moving body
  - Extrinsic rotations(**s**tatic): coordinate axes attached to a static body
- If **s** or **r** is not specified, test it before use

```
# map axes strings to/from tuples of inner axis, parity, repetition, frame
_AXES2TUPLE = {
    'sxyz': (0, 0, 0, 0), 'sxyx': (0, 0, 1, 0), 'sxzy': (0, 1, 0, 0),
    'sxzx': (0, 1, 1, 0), 'syzx': (1, 0, 0, 0), 'syzy': (1, 0, 1, 0),
    'syxz': (1, 1, 0, 0), 'syxy': (1, 1, 1, 0), 'szxy': (2, 0, 0, 0),
    'szxz': (2, 0, 1, 0), 'szyx': (2, 1, 0, 0), 'szyz': (2, 1, 1, 0),
    'rzyx': (0, 0, 0, 1), 'rxyx': (0, 0, 1, 1), 'ryzx': (0, 1, 0, 1),
    'rxzx': (0, 1, 1, 1), 'rxzy': (1, 0, 0, 1), 'ryzy': (1, 0, 1, 1),
    'rzxy': (1, 1, 0, 1), 'ryxy': (1, 1, 1, 1), 'ryxz': (2, 0, 0, 1),
    'rzxz': (2, 0, 1, 1), 'rxyz': (2, 1, 0, 1), 'rzyz': (2, 1, 1, 1)}
```

24 Euler Angle Conventions in **transforms3d**

`pytorch3d.transforms.euler_angles_to_matrix`*(euler_angles: torch.Tensor, convention: str)* → torch.Tensor [source]

Convert rotations given as Euler angles in radians to rotation matrices.

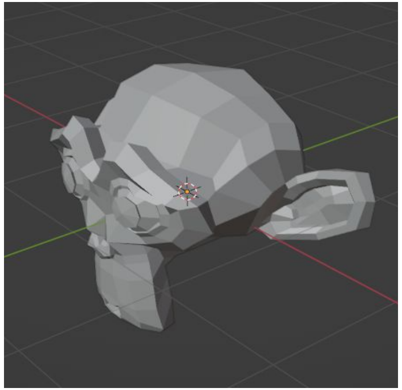
Parameters:
- **euler_angles** – Euler angles in radians as tensor of shape (..., 3).
- **convention** – Convention string of three uppercase letters from {"X", "Y", and "Z"}.

Returns: Rotation matrices as tensor of shape (..., 3, 3).

s or **r** unspecified
Be cautious
pytorch3d

Default Coordinate Frames

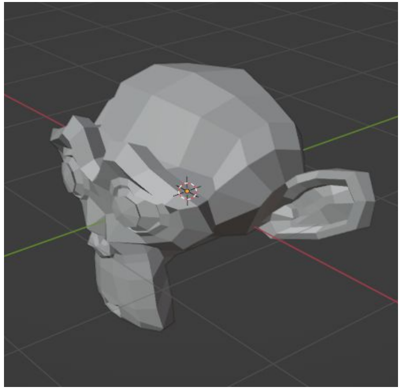
- Objects changes orientation when modeled in Blender, exported as obj, and imported in SAPIEN.

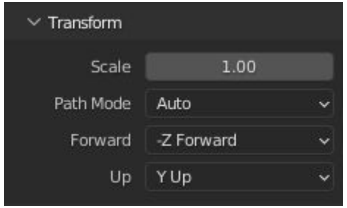


?

Default Coordinate Frames

- Objects changes orientation when modeled in Blender, exported as obj, and imported in SAPIEN.
- Different software and file formats use different coordinate frame conventions.

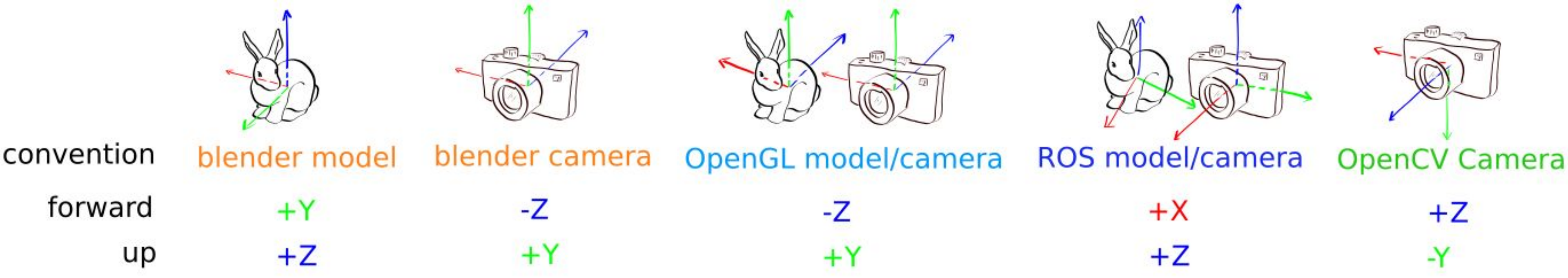




Blender .obj exporter changes the frame by default. SAPIEN does not make frame assumptions based on format.

Default Coordinate Frames

- Objects changes orientation when modeled in Blender, exported as obj, and imported in SAPIEN.
- Different software and file formats use different coordinate frame conventions.



These are common choices, not always true and may be customized.

Default Coordinate Frames

- Objects changes orientation when modeled in Blender, exported as obj, and imported in SAPIEN.
- Different software and file formats use different coordinate frame conventions.

- Tip: visualize and inspect loaded models when using assets from a new source.

Joint Order of Robots

- Even with the same URDF, different software can parse the order of joints in different ways.

- Common Issue:
    a. Train an RL algorithm to control a robot in a simulator.
    b. Action space is defined as joint velocity/position/force.
    c. Deploy the RL policy on a real robot.
    d. Joint order may not match between simulator and real robot.

Outline

- Causes of common bugs: conventions in robotics
- Causes of common bugs: simulation assets
- Causes of common bugs: physical solver
- Causes of common bugs: renderer
- Causes of common bugs: controller
- Environment speed

Causes of common bugs: Simulation Assets

- Gaps between collision and visual mesh
- Collision shapes changed after loading
- Issues in objects with small mass/inertia
- Self-collision from bad modeling
- Issues in empty robot links

Gap Between Collision and Visual Mesh

- Robots often provide 2 types of meshes
    - **Visual**: for rendering only (fancy-looking)
    - **Collision**: for simulation (low-poly, often convex)
    - What you see is not used for collision checking!
    - Run empty.py

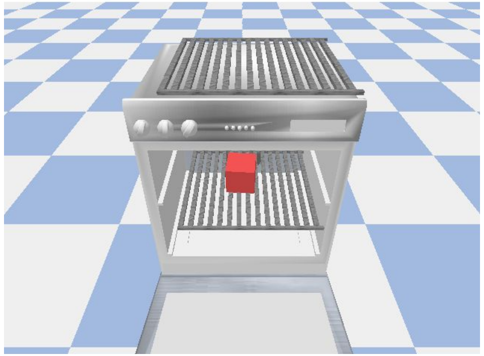
```xml
<link name="panda_link1">
  <visual>
    <geometry>
      <mesh filename="franka_description/meshes/visual/link1.dae"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
      <mesh filename="franka_description/meshes/collision/link1.stl"/>
    </geometry>
  </collision>
</link>
```

Visual

Collision

Collision Shapes
Change After Loading

- Issue posted to SAPIEN Github
    - An oven is loaded in PyBullet
    - A cube is shot out with seemingly no collision
- Can reproduce in SAPIEN (a completely different framework)
    - Run convex.py



Collision Shapes
Change After Loading

- Most simulations require **convex** collision shapes and will take the convex hull of provided collision shapes.
- Solution
    - Use **A**pproximate **C**onvex **D**ecomposition to represent the collision shape.
    - V-HACD is the most choice and is built into PyBullet.
    - Collision-aware ACD developed at our lab preserves detailed structures better.

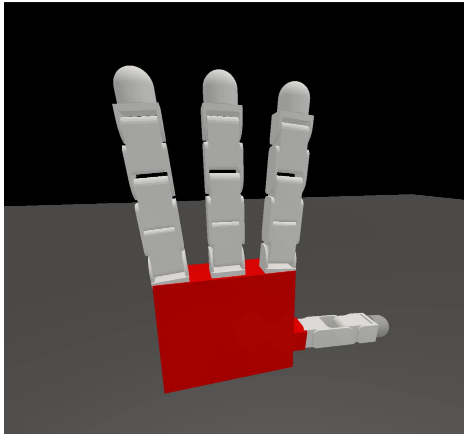
https://github.com/kmammou/v-hacd
https://colin97.github.io/CoACD/

Small Mass/Inertia

- Sometimes, a loaded object does not respond to any applied force/torque
  - If the mass/inertia is too small, the simulation may not be able to simulate it due to floating point error, or simply by design.
  - Run small_mass.py
  - Quick check: mass and inertia should be greater than 1e-7
  - Increase the mass and inertia to see if the issue goes away

Self-Collision from Bad Modeling

- URDF from Github may not be perfect
  - If your algorithm does not work, do not blame it…
  - Maybe the robot model has some problems
  - Run check_urdf.py -u=../assets/allegro_hand_description/allegro_hand.urdf
  - The palm and thumb finger link collide (in red) at initial joint position, leading to unstable motion
  - Check the URDF and resolve undesired self-collisions first



Empty Robot Links

- Empty/dummy link:
  - No geometry are attached
  - Often used as connector between non-empty links
- Empty link may influence robot dynamics
  - Add additional mass/inertia onto the robot
    - E.g. PyBullet gives a warning and set mass to **1(kg)**!
    - It can dominate dynamics when connected links have small mass, e.g. robot finger (~0.01 kg)

```xml
<link name="panda_link8"/>
<joint name="panda_joint8" type="fixed">
  <origin rpy="0 0 0" xyz="0 0 0.107"/>
  <parent link="panda_link7"/>
  <child link="panda_link8"/>
  <axis xyz="0 0 0"/>
</joint>
```

Link8 of the panda robot is an empty link

Outline

- Causes of common bugs: conventions in robotics
- Causes of common bugs: simulation assets
- Causes of common bugs: physical simulator
- Causes of common bugs: renderer
- Causes of common bugs: controller
- Environment speed

Causes of common bugs: Physical Simulator

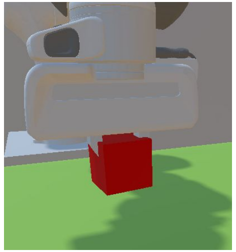
- Simulation reset
- Undesired penetration
- Unstable grasping
- Contact properties

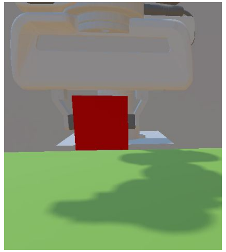
Simulation Reset

- Run reset.py
- Resetting simulation to a previous state
  - Positions
  - Velocities
  - Constraints (e.g. controller parameters, controller targets)
- Simulation is not always deterministic
  - Resetting and replaying may not result in the same outcome
  - Mainly caused by iterative constraint solvers

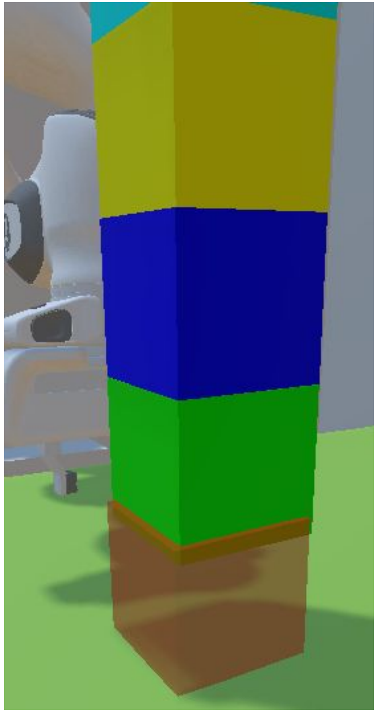
Undesired Penetration

- Time step
  - Run stack.py
  - Taking smaller steps almost always make the solver more stable
  - Smaller steps means slower simulation
- Solver iterations



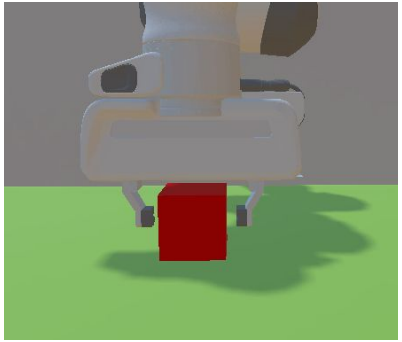


Max solver iterations 2 5



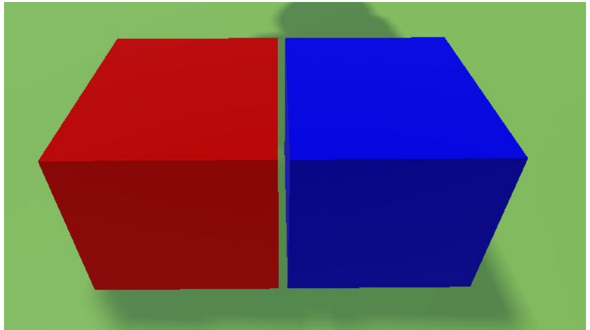
Grasping Stability: Friction and Solver Parameters

- Most likely
  - The block is too heavy and the gripping force and friction coefficient are not large enough
  - Run friction.py
  - Debug method: try to increase the friction, and verify the change.
- Other possible reasons
  - Time step too large
  - Solver iterations too small



Contact Properties

- What is a contact
  - Objects with distance smaller than a threshold
  - Most use cases want contacts with force instead of all contacts
  - E.g. this is a contact



Outline

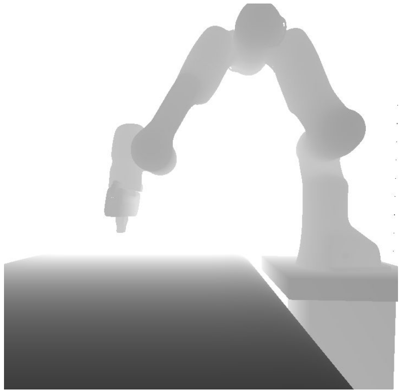
- Causes of common bugs: conventions in robotics
- Causes of common bugs: simulation assets
- Causes of common bugs: physical solver
- Causes of common bugs: renderer
- Causes of common bugs: controller
- Environment speed

Causes of common bugs: Renderer

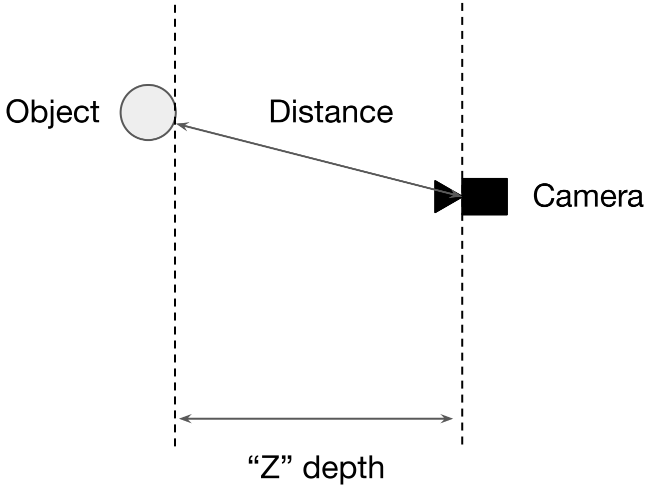
- Definition of depth map (z depth vs distance)
- Renderer depth buffer (z-buffer)
- Depth of transparent objects
- Point cloud from depth
- Matrices in vision and rendering

Depth Map

- Many possible ways to provide the depth map
  - Z depth: distance along the camera axis (most common)
    - May be positive or negative
  - Distance (ray depth): distance along the camera ray



Z-depth, positive



Depth Buffer

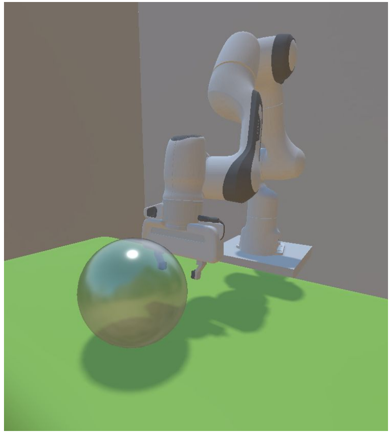
- Many possible ways to provide the depth map
  - Z [linear] depth: distance along the camera axis
  - Z-buffer depth: raw depth from renderer depth buffer
    - Range [0, 1], not linear
    - Convert from z-buffer depth to linear depth

$$z_l = 1/\mathrm{lerp}(1/n, 1/f, z_b) \quad \begin{array}{l} n: \text{ near clip plane} \\ f: \text{ far clip plane} \end{array}$$

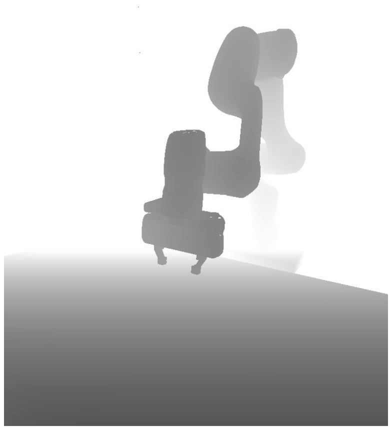
Note: this is the most common choice. There are other z-buffer conventions. Run a test when in doubt.

Depth of Transparent Objects

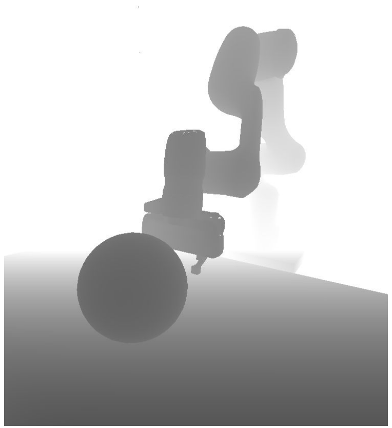
- Should we include or ignore the transparent object?
    - Most environments include the transparent object



RGB



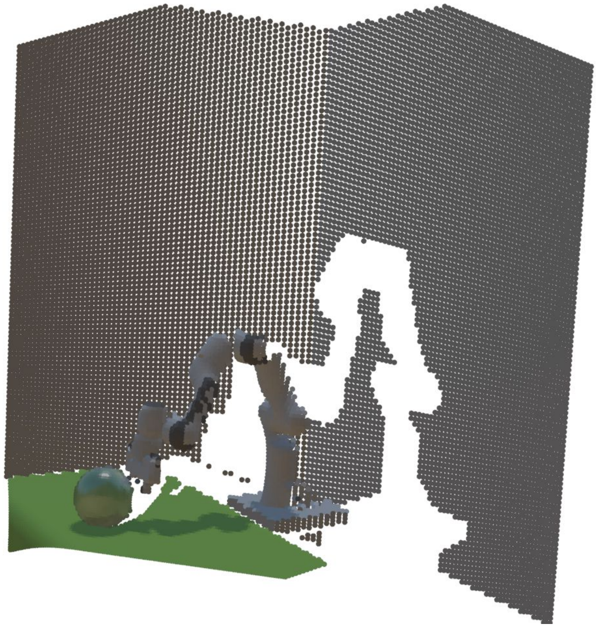
Opaque depth



Transparent depth

Point Cloud From Depth

- Converting depth maps to point clouds is not always easy. (See next slides)
- Tips
  - Look for a built-in API to get point clouds and hope it exists.
  - Visualize and inspect the point clouds with some library, e.g.
    - Trimesh
    - Open3D

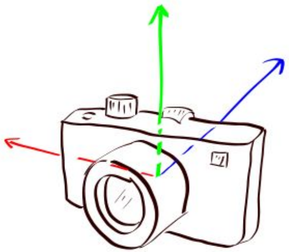
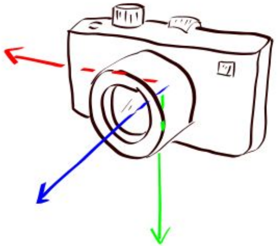


Matrices in Vision and Rendering

- Vision community and graphics community use different matrices to represent the camera
    - Graphics: model matrix, view matrix, projection matrix
    - Vision: extrinsic matrix, intrinsic matrix

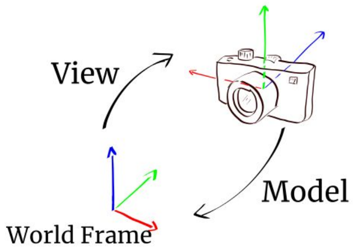
Matrices in Vision and Rendering

- Convention for camera coordinate frame

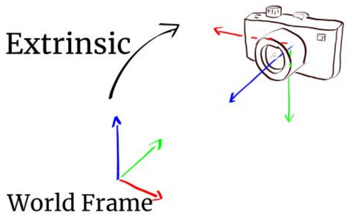
	Rendering/OpenGL	Vision/OpenCV
		
Forward	$-Z$	$+Z$
Upward	$+Y$	$-Y$

Matrices in Vision and Rendering

- **View Matrix** vs **Extrinsic Matrix**
  - Model matrix (4x4): rendering camera pose in world frame
  - View matrix (4x4): inverse of model matrix, transforms points in the world frame to points in the rendering camera frame
  - Extrinsic matrix (3x4): view matrix but in the vision convention



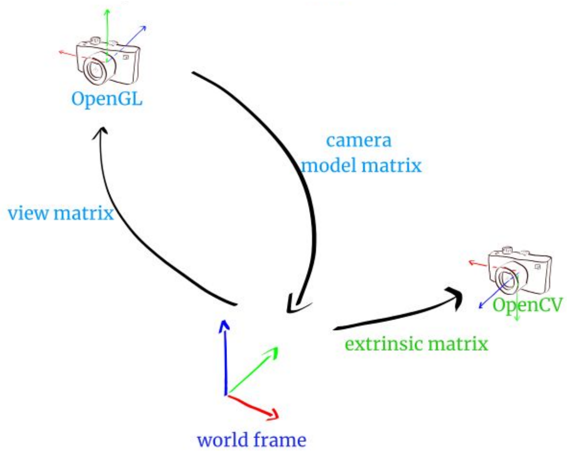
Rendering/OpenGL



Vision/OpenCV

Matrices in Vision and Rendering

- **Projection Matrix** vs **Intrinsic Matrix**

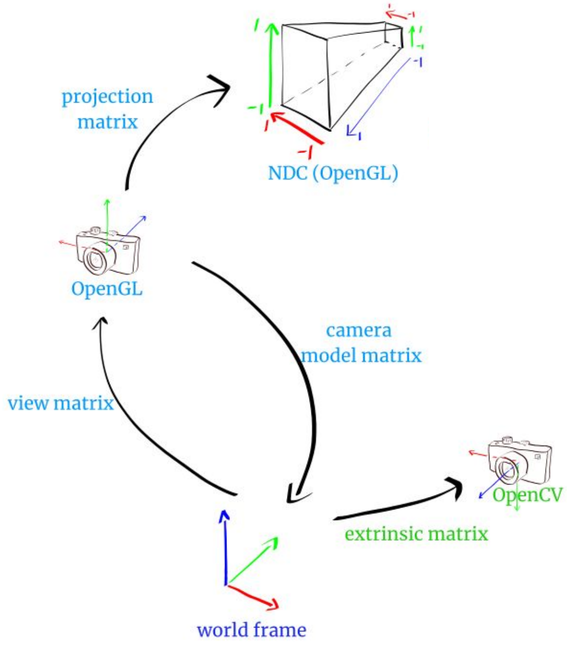


Matrices in Vision and Rendering

- **Projection Matrix** vs **Intrinsic Matrix**

Projection Matrix: project points to normalized device coordinates (NDC).

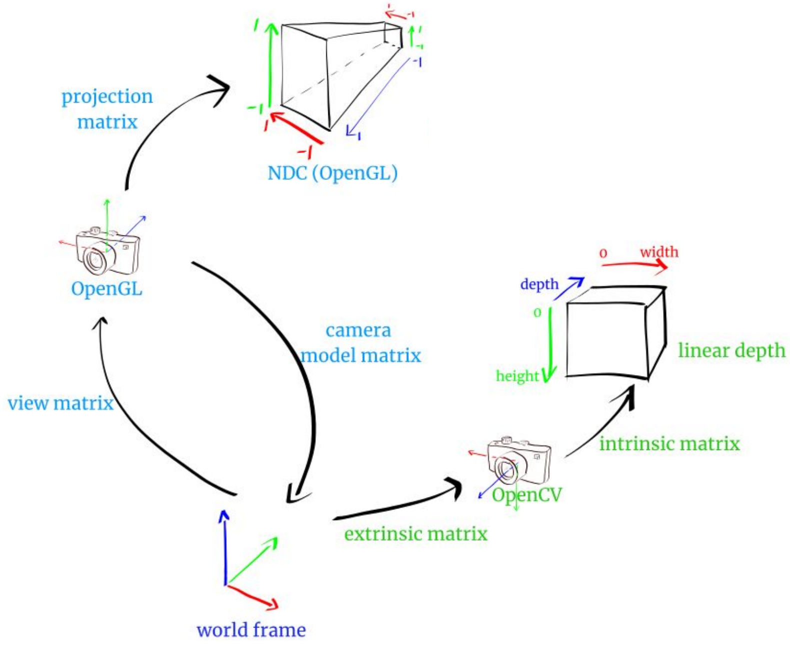
NDC is often a unit cube, sometimes the depth (z-buffer) is in range [0,1] instead of [-1,1].



Matrices in Vision and Rendering

- **Projection Matrix** vs **Intrinsic Matrix**

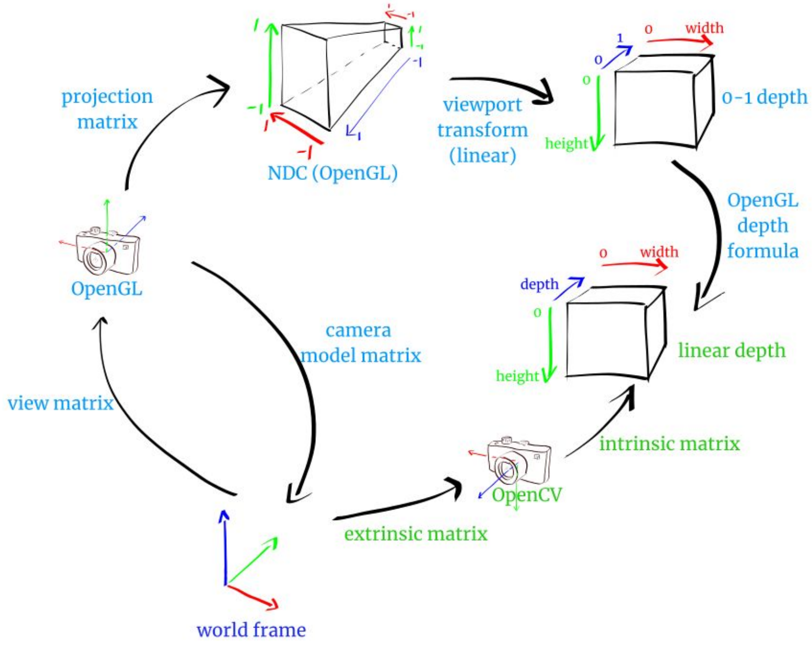
Intrinsic Matrix: project points to image coordinates with linear depth



Matrices in Vision and Rendering

- **Projection Matrix** vs **Intrinsic Matrix**

Connect NDC with image coordinates: a linear "viewport transform" plus a depth conversion.



Matrices in Vision and Rendering

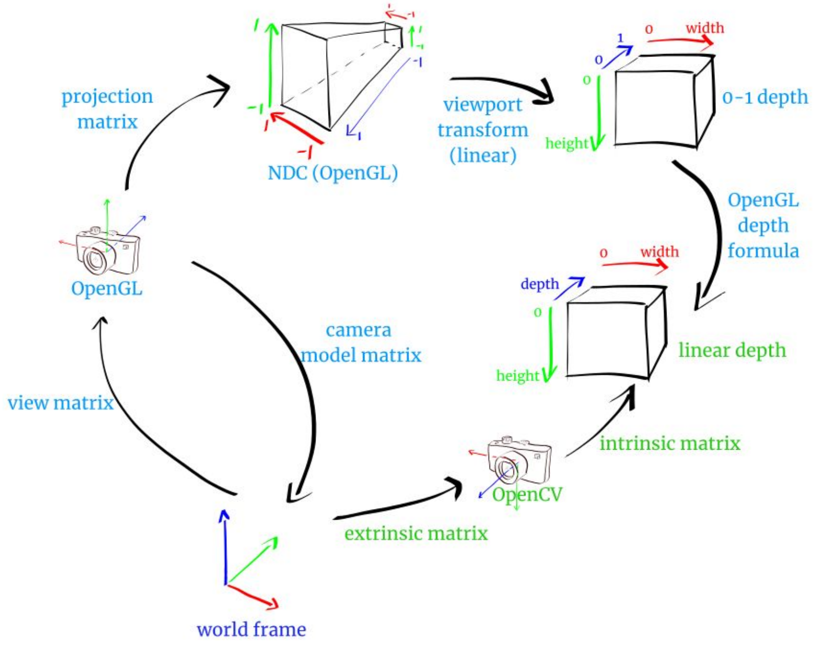
- **Projection Matrix** vs **Intrinsic Matrix**

Projection matrix

$$\begin{bmatrix} \frac{2f_x}{W} & -\frac{2s}{W} & -\frac{2c_x}{W}+1 & 0 \\ 0 & \frac{2f_y}{H} & \frac{2c_y}{H}-1 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Intrinsic matrix

$$\begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

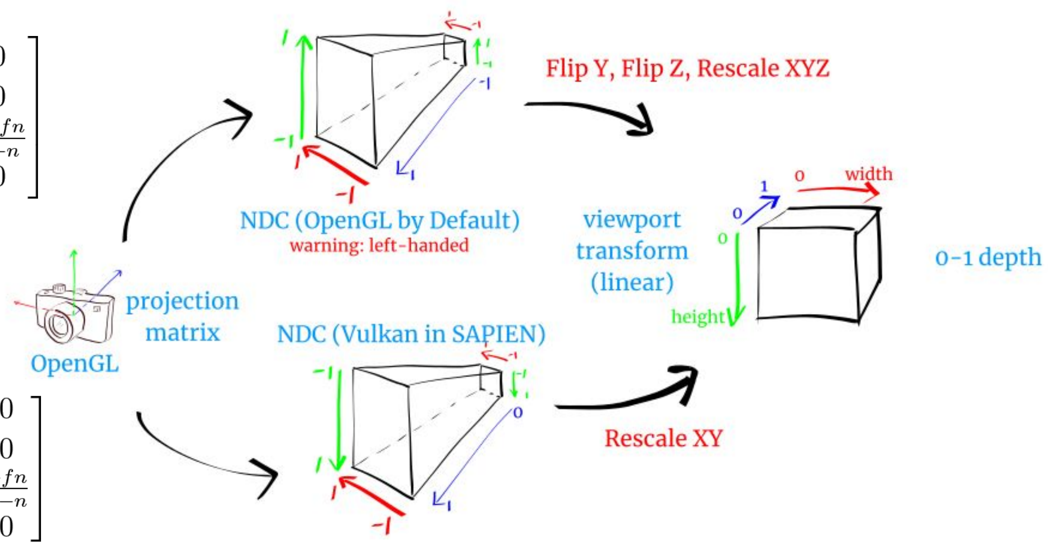


Matrices in Vision and Rendering

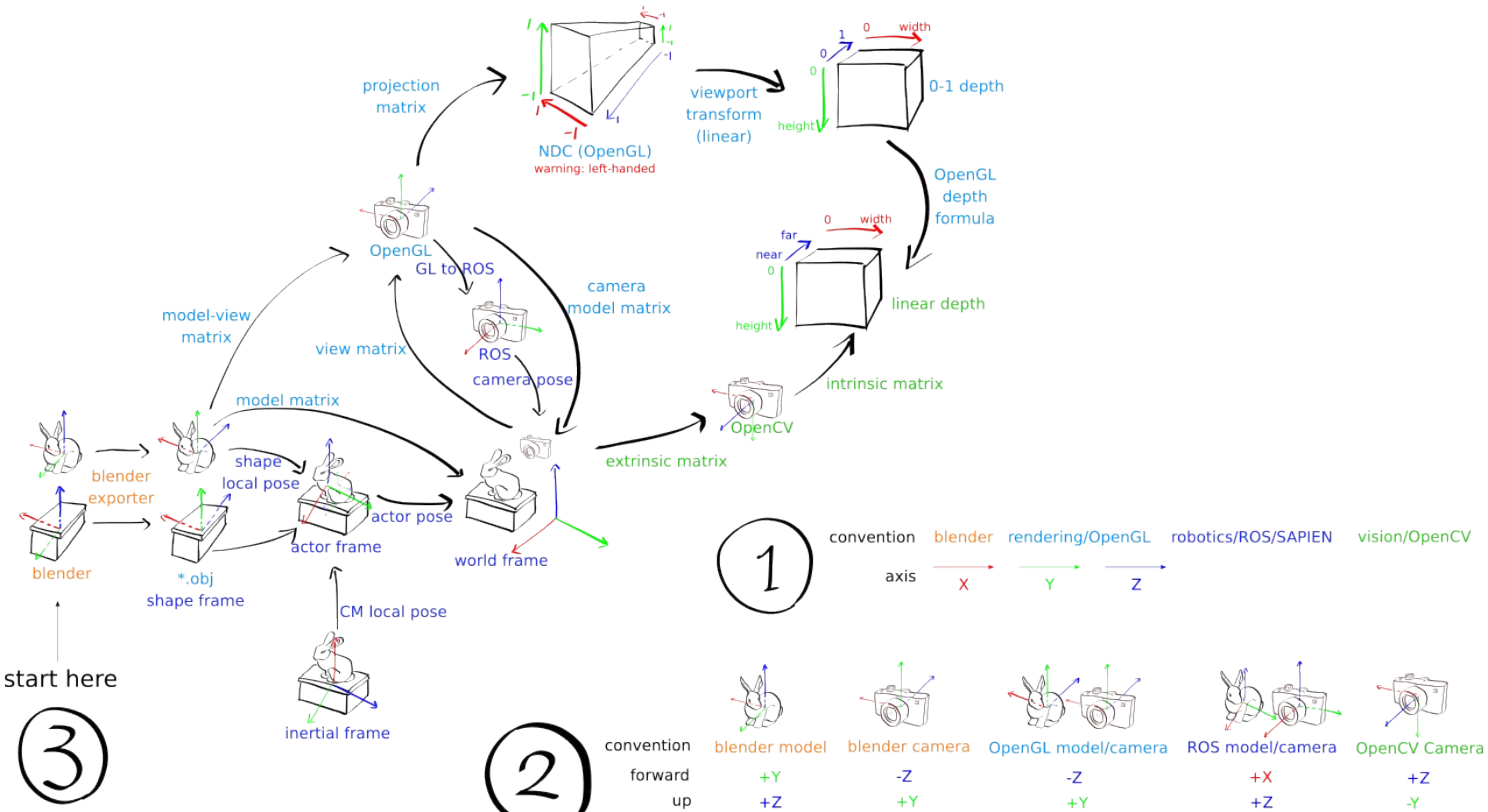
- Different projection matrix conventions
  - Avoid projection matrices whenever possible
  - Perform extensive testing

$$\begin{bmatrix} \frac{2f_x}{W} & -\frac{2s}{W} & -\frac{2c_x}{W}+1 & 0 \\ 0 & \frac{2f_y}{H} & \frac{2c_y}{H}-1 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \frac{2f_x}{W} & -\frac{2s}{W} & -\frac{2c_x}{W}+1 & 0 \\ 0 & -\frac{2f_y}{H} & -\frac{2c_y}{H}+1 & 0 \\ 0 & 0 & \frac{-f}{f-n} & \frac{-fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



Too Many Transformations…



Outline

- Causes of common bugs: conventions in robotics
- Causes of common bugs: simulation assets
- Causes of common bugs: physical solver
- Causes of common bugs: renderer
- Causes of common bugs: controller
- Environment speed

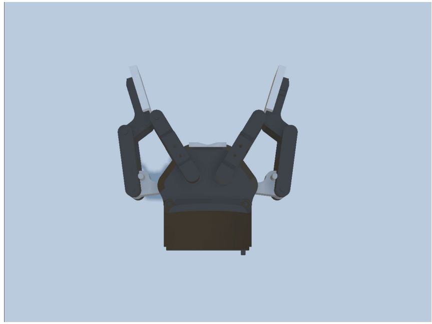
Causes of common bugs: Controller

- Gripper with non-parallel motion: Robotiq Gripper
- Position controller vs “set position”
- Balancing passive force
- Unstable motion of End-Effector(EE) controller
- Joint limits in controller design

Gripper with Non-Parallel Motion

- Some grippers, e.g. Robotiq, has non-parallel motion generated from 6 **inter-dependent** joints
- Direct loading into simulator -> joints are **independent**
- Issue**:** mechanical constraint is not well-modeled in the URDF

Real Robotiq 2F-85



Sim Robotiq 2F-85 without constraint modeling

Gripper with Non-Parallel Motion

- Run robotiq.py -c

- By adding constraints, the motion can be modeled
- However, adding loop constraints also brings instability
- Be cautious when using such tricks

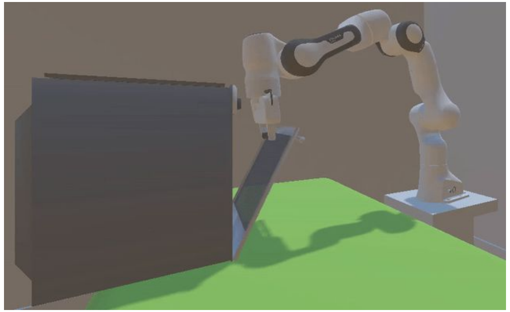
Sim Robotiq 2F-85 with constraint modeling

Balance Passive Force

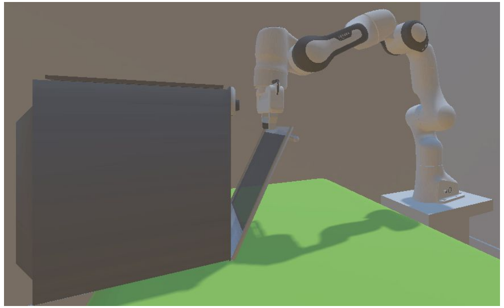
- “My robot never reaches target positions. Are my PD controllers bad?”
- PD controller target is only reached when there are no other forces.
  - Passive forces
    - Gravity
    - Centrifugal and Coriolis force
- **Augmented PD Control**: compute and apply additional joint force/torque to balance passive forces along with PD controllers.

Position Controller vs Set Position

- During dynamics simulation, never **set** position/pose.
- Position controller
  - Compute force/torque
  - Respect physics
- Set position
  - Teleport to configuration
  - Do it no matter what.



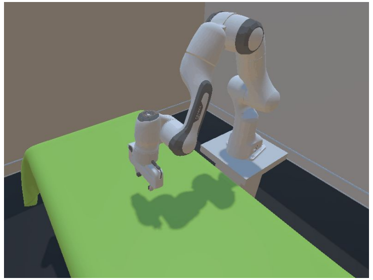
Position Controller



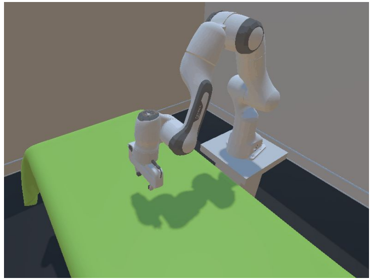
“Set Position”

Unstable Motion of EE Control

- "Why my robot arm is sometimes shaking?"
- IK solving is not stable when close to singularity. Possible solution:
  - Increase the control frequency
  - Increase damping in the IK solver.
- Compare *ee_control.py -d=0.01* and *ee_control.py -d=0.05*



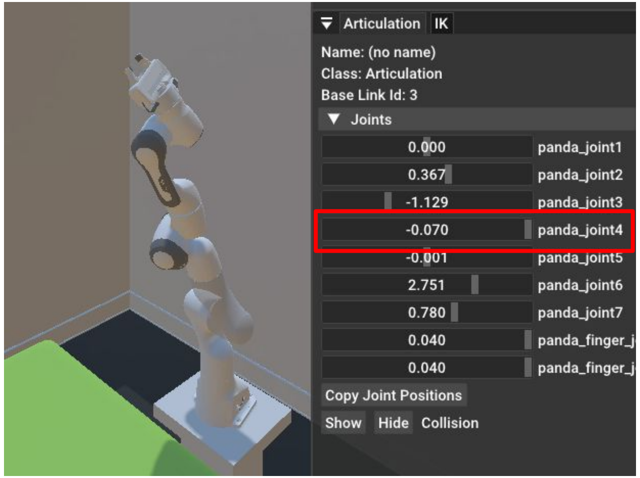
damping=0.01



damping=0.05

Joint Limits in Controller Design

- “My robot end-effector does not move as desired.”
- Most IK solver/EE controller does not consider joint limit
  - Check whether the robot reaches a joint limit when observing unsired controller behavior.
  - Try to avoid reaching joint limits in your algorithm design.



Outline

- Causes of common bugs: conventions in robotics
- Causes of common bugs: simulation assets
- Causes of common bugs: physical solver
- Causes of common bugs: renderer
- Causes of common bugs: controller
- Environment speed

Common Issue: Environment Speed

- Optimizing environment speed is hard
- General guideline
  - Debug in a single process/thread
  - Build a profiler. Profile the following
    - Total time for stepping simulation
    - Total time for rendering functions
    - Total time for expensive planning/network evaluation
    - Other time

Profiler Examples

- Habitat's visual profiler tutorial
  - https://www.youtube.com/watch?v=I4MjX598ZYs&list=PLGywud_-HlCORC0c4uj97oppQrGiB6JNy
  - Py-spy for Python code
  - Nsight for CUDA
  - Their approaches can be applied to any other python-based environments

Rendering Speed

- Rendering is the bottleneck
  - Check your loaded meshes
    - Are there meshes with millions of triangles?
  - Check number of objects
  - Switch to a lighter renderer
    - If you do not need RGB, switch to a depth-only renderer can save time and memory

Physical Simulation Speed

- Physical simulation is the bottleneck
  - If single step is consistently slow
    - Check whether there is undesired collision.
    - Inspect number of objects in the scene.
    - Are there objects with very complex collision?
  - If the time for a single step varies
    - It is typically slow when there are a lot of collisions
    - Disable unnecessary collision checking may help

Summary

- Conventions in robotics
- Simulation assets
- Physical simulator
- Renderer
- Controller
- Environment speed